uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática

Curso Académico 2017-2018

Trabajo Fin de Grado

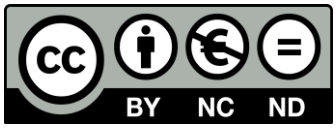
"Detección de Dedos Utilizando Técnicas de Visión por Computador y Machine Learning"

Fernando Martín Rivas

Tutor

Juan Miguel García Haro

Octubre 2018



RESUMEN

El objetivo del trabajo consiste en realizar un algoritmo capaz de contar el número de dedos que está mostrando una mano aplicando técnicas de visión artificial y Machine Learning.

Para la resolución del problema se ha utilizado una cámara infrarroja con la tecnología de luz estructurada para la captación de imágenes. En el apartado de Machine Learning se han comparado 4 métodos distintos para la identificación de las imágenes de la mano. Los métodos estudiados han sido Maquinas de Vectores de Soporte, Bosques Aleatorios, K-Vecinos más Cercanos y Perceptrón Multicapa. Los tres primeros se corresponden con técnicas de Machine Learning tradicional, mientras que el último entra dentro del campo del Deep Learning. También se han estudiado los conceptos básicos de la visión artificial necesarios para llevar a cabo este trabajo.

Este trabajo también puede ser considerado como un tutorial para adentrarse en el mundo de la inteligencia artificial puesto parte desde cero. El proceso comienza con la creación de una base de datos, incluyendo el filtrado completo de la imagen, y finaliza con el análisis de los resultados obtenidos tras realizar distintas pruebas.

El método que mejor ha trabajado utilizando nuestra propia base de datos ha sido el de Máquinas de Vectores de Soporte, tanto en el análisis utilizando las imágenes de la base de datos no utilizadas en el entrenamiento, como en el análisis en tiempo real, consiguiendo más de un 95% de acierto en el primer caso.

Palabras clave: Machine Learning, Visión Artificial, Máquinas de Vectores de Soporte, K-Vecino más Cercano, Bosques Aleatorios, Perceptrón Multicapa.

ABSTRACT

The aim of this work is to create an algorithm that count the number of fingers that is showing a hand, by applying artificial vision and machine learning techniques.

To get the images, an infrared camera with structured light technology has been used. In the machine learning section, 4 different methods had been compared in order to identify the images of the hand. The analysed methods are Support Vector Machines, Random Forest, K-Nearest Neighbour, and Multilayer Perceptron. The first three correspond with traditional machine learning algorithms, while the last one is in the deep learning field. The necessary basic concepts of artificial vision needed to carry out this project are also studied.

This work could be also a tutorial to get inside the artificial intelligence world as it starts from the scratch. The process begins with the creation of a database, including all the filtering of the images, and ends with the analysis of the obtained results before doing different trials.

The method with the best results by using our own database is Support Vector Machines, both in the analysis with the image in the database that where not used in the training as in the real time analysis, obtaining more than a 95% of right guess in the first case.

Key words: Machine Learning, Artificial Vision, Support Vector Machines, K-Nearest Neighbour, Random Forest, Multilayer Perceptron.

INDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

1. INTRODUCCION

1.1 Motivación

El mundo de la robótica cada vez está más presente en nuestro alrededor. Es por eso por lo que la interacción entre seres humanos y máquinas cada vez deberá ser más sencilla para que esta tecnología pueda estar al alcance de cualquier persona sin la necesidad de que deban tener unos conocimientos avanzados sobre programación o robótica. Ahí es donde entra la idea de la visión artificial. La visión artificial, junto con el reconocimiento de voz, son los dos medios más utilizados para lograr que pueda haber una comunicación simple con las máquinas. Contar dedos solamente es solo uno de los muchos objetivos que se pueden lograr mediante el uso de la visión artificial.

Otro de los avances que ha ayudado de gran manera al descomunal desarrollo que ha sufrido la robótica en los últimos años, es el uso del Machine Learning, un término que cada vez es más escuchado en los entornos de programación, aunque en muchas ocasiones es un gran desconocido.

En este trabajo se ha intentado hacer una unión de estas dos técnicas, la visión artificial y el Machine Learning, para crear un algoritmo que pueda aportar un poco más al mundo de la interacción entre el hombre y la máquina.

1.2 Objetivos

El objetivo principal del trabajo es el de crear un mecanismo capaz de contar los dedos de una mano humana a tiempo real aplicando técnicas de Machine Learning y con la ayuda de una cámara de luz estructurada. Para lograr este objetivo nos hemos marcado una serie de sub objetivos más concretos:

- El primer objetivo que se nos plantea es el de conseguir una buena base de datos lo suficientemente buena como para que nuestro clasificador pueda trabajar de forma correcta. En primer lugar, se nos planteaba la idea de utilizar una base de datos ya existente en línea, pero al no encontrar ninguna convincente que se pudiese ajustar a nuestro trabajo, decidimos crear nuestra propia base de datos.

- Para lograr el objetivo anterior, nos aparece un nuevo objetivo, el cual es crear la base de datos. Para conseguir esto, primero deberemos entender de forma correcta la teoría de la visión artificial y el funcionamiento de una cámara de luz estructurada.
- Una vez tenemos una base teórica consistente, deberemos conseguir un correcto filtrado de las imágenes. Además, es interesante que este filtrado funcione en tiempo real para que más adelante, se puedan realizar predicciones dinámicas con las imágenes filtradas al momento.
- Cuando hayamos conseguido unas imágenes que consideremos que se puedan utilizar para tareas de Machine Learning, deberemos estudiar cómo trabajan los algoritmos de Machine Learning con Python y con las librerías que vamos a utilizar, para poder estructurar y transformar nuestras imágenes en datos válidos para los clasificadores.
- Una vez visto esto, estudiaremos los distintos métodos de Machine Learning que existen aplicados a tareas de visión artificial para ver cuál puede trabajar mejor en nuestra aplicación. En nuestro caso se han elegido 4 de los algoritmos más importantes para llevar a cabo una comparación entre ellos.
- Antes de llevar a cabo las predicciones, deberemos buscar los parámetros que hagan que nuestros clasificadores trabajen de la mejor forma posible.
- Uno de los últimos objetivos será analizar los resultados obtenidos una vez hayamos encontrado los mejores parámetros, y deberemos ver si se han conseguido unos porcentajes de acierto lo suficientemente altos como para decir que nuestro clasificador funciona correctamente.
- Por último, nuestro objetivo final es el de trasladar el algoritmo de clasificación a una aplicación en tiempo real, la cual sea capaz como ya habíamos dicho, de decir cuántos dedos está mostrando la mano en cada momento.

1.3 Marco socioeconómico

En este apartado analizamos de forma superficial el impacto tanto social como económico que tienen las aplicaciones de Machine Learning en el día a día. La inteligencia artificial está cada vez más implementada en nuestra sociedad casi sin que nos demos cuenta.

Esta tecnología ha conseguido que se puedan realizar acciones que hace pocos años eran casi inimaginables. Desde coches que circulan sin la necesidad de intervención de ningún ser humano y aplicaciones de reconocimiento por voz, hasta asistirnos a la hora de elegir qué película ver basándose en otras que hayamos visto como por ejemplo hace Netflix [1].

El Machine Learning también tiene un gran impacto en el entorno sanitario. Se están empezando a crear algoritmos que son capaces de diagnosticar enfermedades en un paciente a partir de los síntomas que este tenga, mediante la obtención de una serie de patrones que pueden ser imperceptibles para un ser humano, pero no para una máquina.

El avance de este tipo de técnicas también supone un gran avance a nivel laboral. La inteligencia artificial se puede aplicar a la hora de optimizar cualquier tipo de proceso industrial, reduciendo enormemente su coste. Además, poco a poco están apareciendo profesiones en las que se puede sustituir fácilmente la mano de obra humana por la de una máquina. Las ventajas de este cambio entre muchas otras son, por ejemplo, que una máquina puede trabajar durante las 24 horas del día, no enferma ni tiene vacaciones y carece de sindicatos. Todo esto tiene una parte negativa, y es que reduce muchísimos puestos de trabajo, haciendo que mucha gente tenga que ser despedida y buscar otras alternativas de empleo. La consultoría Accenture, por ejemplo, determinó en 2017 que 17000 puestos de trabajo que en ese momento ocupaban personas, iban a ser reemplazados por maquinas gracias a la automatización [2]. En este caso particular, todas las personas fueron reasignadas a distintos puestos de trabajo dentro de la empresa para que no perdiesen sus empleos, pero esto no siempre es posible.

Cabe decir que el hecho de que cada vez haya más empleos que se puedan sustituir por maquinas automatizadas, no quiere decir que el número total de empleos disminuya. Cada día están apareciendo empleos nuevos relacionados con el Machine Learning, y en un futuro habrá algunos empleos que ni siquiera nos imaginemos que puedan existir. Es por eso por lo que la educación tiene un papel fundamental para que estas tecnologías se puedan aplicar de una manera equilibrada y lógica. Poco a poco habrá que ir

adaptándose a una nueva era en la que las maquinas vayan cogiendo más terreno tanto en el mundo social como laboral.

Como conclusión podemos decir que toda esta automatización es realmente útil, y nos facilita la vida tanto en nuestro día a día como en nuestra vida laboral, siempre y cuando esta evolucione de una forma ética y responsable, empezando por una educación eficiente sobre la materia.

1.4 Marco regulador

Hoy en día no existen leyes que regulen aplicaciones que utilicen inteligencia artificial, ni de forma nacional ni internacional, aunque si es cierto que la mejora progresiva de este tipo de tecnologías genera la obligación o necesidad de crear algún tipo de regulación en un futuro cercano.

El objetivo principal de la inteligencia artificial es el diseñar una máquina que trabaje de la forma más autónoma posible, intentando emular el comportamiento del ser humano. Es por esto por lo que aparece la idea de plantearse hasta qué punto de autonomía se puede dotar a una máquina, algo que podría ser regulado.

Cabe anotar que hoy en día no existen en el mercado maquinas lo suficientemente autónomas como para que puedan suponer un problema ético o moral que deba ser controlado, por mucho que la ciencia ficción a veces nos quiera hacer creer lo contrario en películas taquilleras como “Yo Robot” o “Ex Machina”.

1.5 Estructura del trabajo

- En el primer capítulo realizamos una pequeña introducción en la cual definimos la motivación y los objetivos, así como encuadramos el trabajo dentro de un marco socioeconómico y regulador.
- En segundo lugar, enunciaremos el software y el hardware utilizados en la elaboración del proyecto y haremos una pequeña elaboración del presupuesto.

- En el tercer capítulo nos adentraremos ya en una parte más teórica enfocada en adentrarnos en las técnicas de Machine Learning. Empezaremos contextualizando el trabajo comparándolo con otros trabajos que puedan ser similares y que nos hayan podido servir de inspiración. Pasaremos después a definir los dos grupos en los que se dividen la mayoría de los métodos, para finalmente hacer una pequeña definición de cada uno de los métodos que hemos utilizado individualmente. Primero veremos el método de bosques aleatorios, continuaremos por el de máquinas de vectores de soporte para seguir con K-vecino más cercano, y finalizar con la teoría del perceptrón multicapa.
- El cuarto capítulo, también de carácter teórico, se encargará de definirnos que entendemos por visión por computador, y que debemos saber para poder realizar un trabajo de estas características. Al igual que en el capítulo anterior, comenzaremos enumerando distintos trabajos realizados relacionados con esta materia.
- Una vez hayamos asentado unas bases teóricas, en el capítulo 5 pasaremos a la descripción de todos los pasos que hemos tenido que realizar para la obtención de nuestro algoritmo. Comenzaremos definiendo como se ha realizado el filtrado y obtención de las imágenes con las cuales hemos creado nuestro dataset y de cómo hemos estructurado estas imágenes para poder utilizarlas más adelante. Una vez visto esto, pasaremos a describir como se realiza un entrenamiento, incluyendo los métodos que se han utilizado para la obtención de los distintos parámetros. Seguidamente explicaremos cuales son las distintas técnicas de análisis de resultados que nos serán útiles más adelante para visualizar de forma clara los resultados obtenidos. Por ultimo veremos cómo se ha creado la predicción de los dedos de la mano en tiempo real, así como un pequeño juego de pares y nones.
- El capítulo sexto es uno de los capítulos de mayor relevancia, ya que aquí es donde analizaremos los distintos métodos para poder conseguir los parámetros que mejor trabajen en nuestra aplicación y realizaremos un análisis de los resultados. En primer lugar, analizaremos los resultados por cada método de forma individual, aplicando la predicción sobre datos estáticos que tenemos en nuestra base de datos y que no hemos utilizado para el entrenamiento. Seguidamente analizaremos en conjunto todos los resultados para comparar que

método es el más fiable. Finalmente analizaremos también la predicción en tiempo real, y compararemos los resultados con cada método para ver cual trabaja mejor.

- Finalmente tendremos el capítulo siete en el que analizaremos los objetivos cumplidos y daremos unas conclusiones. También hablaremos de posibles mejoras y trabajos futuros que se puedan hacer partiendo del nuestro.

2 HARDWARE Y SOFTWARE UTILIZADO

2.1 Hardware

Para la elaboración del proyecto se ha utilizado un ordenador portátil de la marca $Lenovo^{TM}$ modelo $Ideapad^{TM}$ Y700 cuyas especificaciones técnicas más relevantes son:

- 16GB de memoria RAM DDR4.
- Procesador $Intel^{®}\ Quad-Core^{TM}$ i7 de sexta generación.
- Tarjeta gráfica dedicada $NVIDIA^{®}$ GTX 960M 4GB GDDR5.

Fig. 2.1. Ordenador Portátil Lenovo Ideapad Y700

La cámara utilizada para la toma de datos es de la marca $Intel^{®}$ modelo $RealSense^{TM}$ F200, la cual venia incorporada de serie en el portátil. Esta cámara utiliza la tecnología de luz estructurada para la captación de profundidad, utilizando como medio la luz infrarroja, y tiene un rango de profundidad de 0.2m a 1.2m, aunque su rango óptimo es de 0.2m a 0.85m.

Fig. 2.2. Cámara Intel RealSense F200

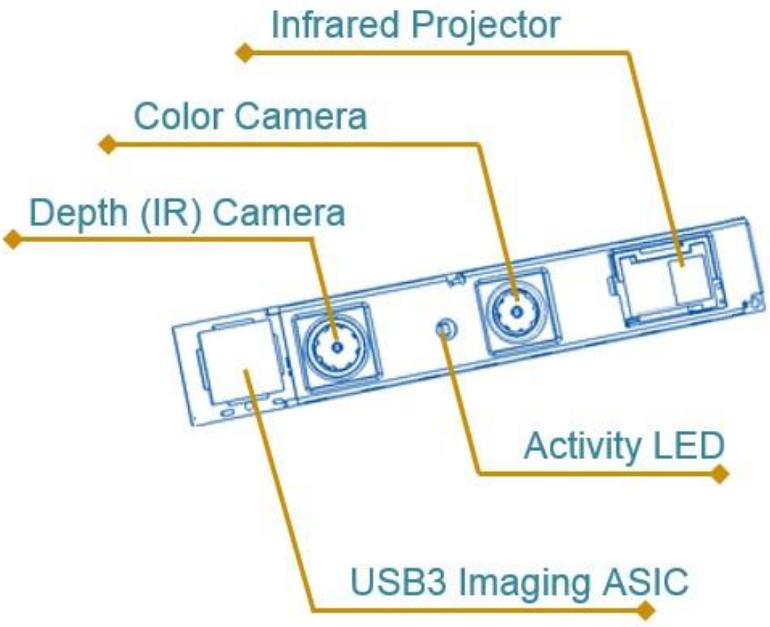


Fig. 2.3. Esquema Cámara Intel RealSense F200

2.2 Software

El sistema operativo con el que se ha trabajado es la distribución de Linux Ubuntu en su versión 16.04 LTS.

El lenguaje de programación con el que se ha realizado el algoritmo, así como todas las pruebas es Python en su versión 2.7.14.

Fig. 2.4. Logotipo de Python

El entorno de programación en el cual se ha compilado todo el código ha sido Jupyter Notebook.

Las librerías de Python utilizadas han sido principalmente:

- NumPy: Utilizada para poder trabajar de forma más cómoda con vectores y con matrices.
- Matplotlib: Utilizada para la representación de gráficas al estilo Matlab.
- OpenCV2: Utilizada para el tratamiento de imágenes digitales
- Scikit-learn: Utilizada para la aplicación de los algoritmos de inteligencia artificial.

2.3 Presupuesto

En este apartado detallamos el presupuesto utilizado para la elaboración del proyecto.

Todo el Software que se ha utilizado para la realización de este proyecto es de carácter gratuito. Tanto el sistema operativo, como las librerías utilizadas son de código abierto a nivel académico y comercial.

En la Tabla [1] podemos ver un desglose de todos los componentes que hemos valorado para la realización de un presupuesto.

Concepto	Multiplicador	Unidades	Importe
Hardware			
Ordenador con cámara	N/A	1	1.526,40 €
Software			
Ubuntu 16.04 LTS	N/A	1	0€
Python 2.7.14	N/A	1	0€
Jupyter Notebook	N/A	1	0€
NumPy	N/A	1	0€
Matplotlib	N/A	1	0€
OpenCV2	N/A	1	0€
Scikit-learn	N/A	1	0€
Mano de Obra			
Búsqueda de información	35€/h	100	3.500,00 €
Generación del algoritmo	50€/h	150	7.500,00 €
Redacción de la memoria	40€/h	120	4.800,00 €
Supervisión del tutor	60€/h	50	3.000,00 €
TOTAL			20.326,40 €

Tabla 1 Presupuesto

Este presupuesto no sería del todo real, ya que el ordenador utilizado está totalmente sobredimensionado con respecto a los requisitos de la aplicación. Este trabajo también podría ser realizado por un ordenador con un valor por debajo de los 500€ junto con una cámara de luz estructurada como puede ser, por ejemplo, la Kinect, fabricada por Microsoft, con un valor estimado de 80€.

3 MACHINE LEARNING

El Machine Learning consiste en conseguir que una maquina aprenda. Tiene como objetivo que, tras realizar un entrenamiento con un determinado tipo de datos, el ordenador sea capaz de clasificar y organizar nuevos datos que le introduzcamos a través de la generalización.

Existen muchos estudios en los cuales no se utilizan técnicas de Machine Learning para la extracción de la mano como pueden ser los estudios realizados por Lal Raheja, Das y Chaudhary [3], en el que utilizan el histograma de la silueta de la mano para la detección de las puntas de los dedos o como por ejemplo el trabajo realizado por Bhuyan, Neog y Kar [4] en el que utilizan la geometría de la mano para la obtención de las puntas de los dedos. A parte de estos trabajos existen muchos más como pueden ser [5] [6] [7] [8].

En nuestro caso hemos decido utilizar técnicas de Machine Learning para obtener nuestro objetivo ya que nos pueden dar unos resultados muchos más versátiles que por la forma geométrica. Existen multitud de trabajos que utilizan la técnica de máquinas de vectores de soporte, como por ejemplo los trabajos de Michahial [9] y el trabajo de Rahman y Afrin [10]. Ambos de estos trabajos utilizan el algoritmo de Canny para la extracción del contorno de la mano, el cual resulta muy útil. Otros muchos trabajos también utilizan los denominado métodos de conjunto como pueden ser los arboles de decisión o los bosques aleatorios en publicaciones como la de Mackie o McCane [11], en la hacen una segmentación de la imagen para luego aplicar el algoritmo de los árboles de decisión, o en el trabajo de P. Li, Ling, X. Li y Liao [12]. Aunque no tan común como en estos dos casos anteriores, en los que vemos trabajos utilizando las técnicas de máquinas de vectores de soporte y bosques aleatorios, también encontramos trabajos que utilizan el algoritmo del k-vecino más cercano. Un ejemplo de este trabajo es el realizado por Nimbalkar, Karhe y Patil [13]. El último método que hemos estudiado es el del perceptrón multicapa, el cual pertenece al campo de las redes neuronales. Por el contrario que con el clasificador del k-vecino más cercano, para este método existen una gran cantidad de referencias en las que nos hemos podido basar para

nuestro estudio como es por ejemplo el trabajo de Vásquez [14], en el que utiliza redes neuronales para contabilizar cuantos dedos está mostrando la mano. El problema que vimos en este trabajo es que utiliza ciertas aplicaciones para el uso de las redes neuronales las cuales requieren de mucha potencia computacional. Estas herramientas son, por ejemplo, Keras o Tensorflow, herramientas muy utilizadas para aplicaciones de reconocimiento de imagen, las cuales hemos querido evitar ya que creemos que nuestra aplicación es demasiado sencilla como para tener que utilizar estas potentes herramientas. Otro ejemplo muy interesante que aplica algoritmos de redes neuronales es el publicado por Xu [15], en el que además de realizar una estimación de la pose, mezclando redes neuronales junto con métodos geométricos como es el teorema de la envolvente convexa, aplica los resultados a sistemas de interacción ordenador-humano.

También ha sido muy útil para la realización de este proyecto fijarnos en trabajos en los que se aplican técnicas de Machine Learning sobre la base de datos MNIST [16], en la cual se encuentran más de 80000 imágenes etiquetadas de números escritos a mano, y que ha sido utilizada por gran cantidad de investigadores del campo de la inteligencia artificial. Algunos ejemplos de trabajos en los que utilizan esta base de datos son [17] [18] [19] y [20] en los que se aplican las técnicas de k-vecino más cercano, bosques aleatorios, máquinas de vectores de soporte y perceptrón multicapa respectivamente.

3.1 Clases de Machine Learning según la base de datos

Existen infinidad de métodos diferentes para realizar tareas relacionadas con la inteligencia artificial. Es por esto por lo que es muy importante saber que métodos deberemos utilizar a la hora de buscar una solución a nuestro problema. Existen dos grandes grupos dentro de los cuales podemos categorizar prácticamente todos los algoritmos de aprendizaje computacional basándonos en los datos que tengamos a priori.

3.1.1 Aprendizaje supervisado

El aprendizaje supervisado tiene lugar cuando poseemos información acerca de los datos que se encuentran en nuestra base de datos. Esto quiere decir que trabajamos con datos etiquetados. Cada dato tiene una etiqueta la cual nos ayudará a saber a qué clase

pertenece. Lo que haremos por lo tanto es enseñarle a este algoritmo cual es el resultado que queremos obtener para un determinado valor de entrada. Tras mostrarle muchos ejemplos, el algoritmo deberá ser capaz de identificar un nuevo valor que no haya visto anteriormente.

Existen una gran cantidad de ejemplos de aplicación de sistemas de entrenamiento que utilizan aprendizaje supervisado y que están presente en nuestro día a día. El ejemplo más claro y más utilizado, es el filtro de spam en nuestro correo electrónico. Este filtro lo que hace es detectar si un correo electrónico que nos acaba de llegar es spam, entendiendo como spam cualquier correo electrónico no solicitado, el cual se ha enviado a un gran número de personas y por lo general tiene fines publicitarios, los cuales la gran mayoría de las veces no nos interesan. En la mayoría de los gestores de correo que se utilizan, existe la posibilidad de marcar manualmente un correo como spam, o no deseado cuando lo abrimos. Esto hará que se guarde esa información, y que cada vez que se reciba un correo con características comunes a este, se mueva automáticamente a la carpeta de spam. Es por esta razón que a medida que avanza el tiempo, este filtro va a ir mejorando hasta llega a un nivel de precisión muy avanzado, ya que su base de datos no para de crecer.

El problema que tiene este tipo de aprendizaje es que puede requerir mucho trabajo a la hora de obtener las etiquetas. Algunas bases de datos constan de miles de datos y muchas veces la única forma de hacerlo es con el ser humano, etiquetando los datos uno por uno.

A su vez, dentro del aprendizaje supervisado, podemos clasificar los métodos de aprendizaje en otros dos grupos. En este caso, la elección del grupo dependerá del resultado que queremos obtener:

- Clasificación: El objetivo de un algoritmo de clasificación es el de dar como resultado una de las etiquetas disponibles a uno de los datos de entrada que no se haya visto con anterioridad

- Regresión: Por otro lado, la regresión nos da como resultado un valor numérico.

En la Figura 3.1 podemos ver un ejemplo claro de la diferencia entre clasificación y regresión dentro de los algoritmos de aprendizaje supervisado.

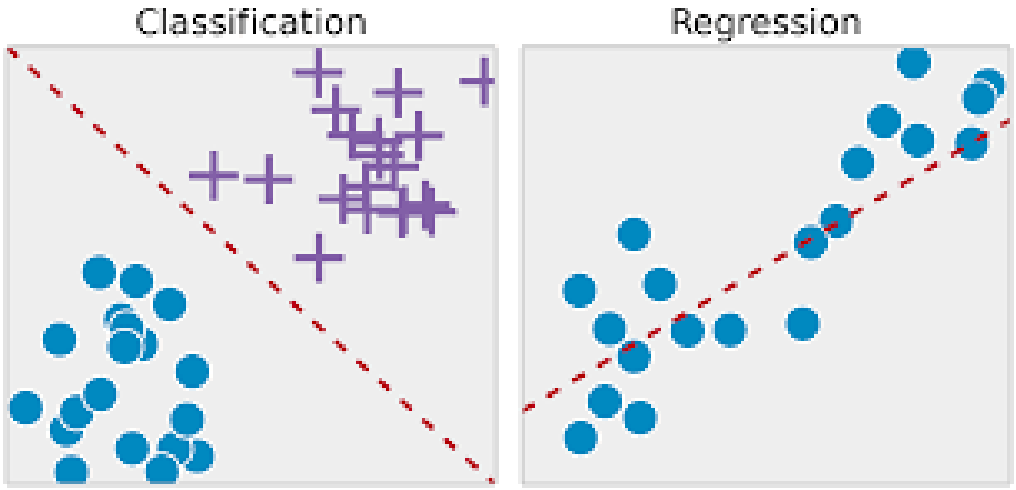


Fig. 3.1 Comparación gráfica entre clasificación y regresión [21]

3.1.2 Aprendizaje no supervisado

En este caso, los datos que proporcionamos a nuestro clasificador no tienen ningún tipo de etiqueta. Lo que hará entonces nuestro algoritmo es buscar similitudes entre todos los datos de entrada. Este tipo de entrenamiento es bastante mas complejo, ya que no conocemos ni los datos de entrada ni los datos de salida.

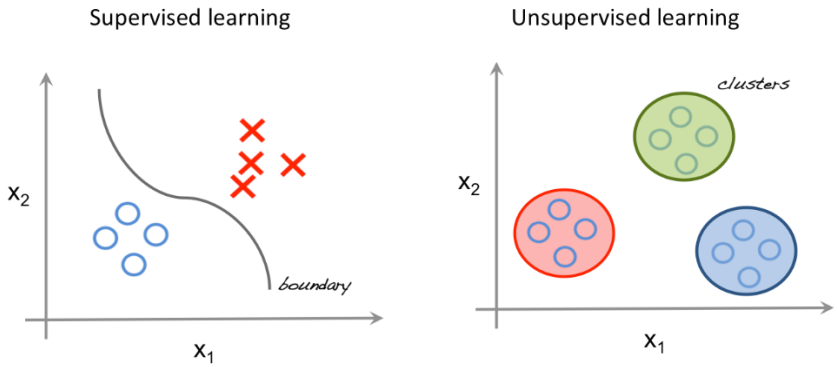


Fig. 3.2 Comparación gráfica entre aprendizaje supervisado y no supervisado [22]

Los dos grupos vistos hasta el momento son con diferencia los más utilizados en tareas de Machine Learning, y conociendo estos dos solamente, podremos ser capaces de resolver más del 90 por ciento de los problemas que se nos pongan por delante. En la Figura 3.2 podemos ver una comparacion grafica entre los dos tipos de mecanismos.

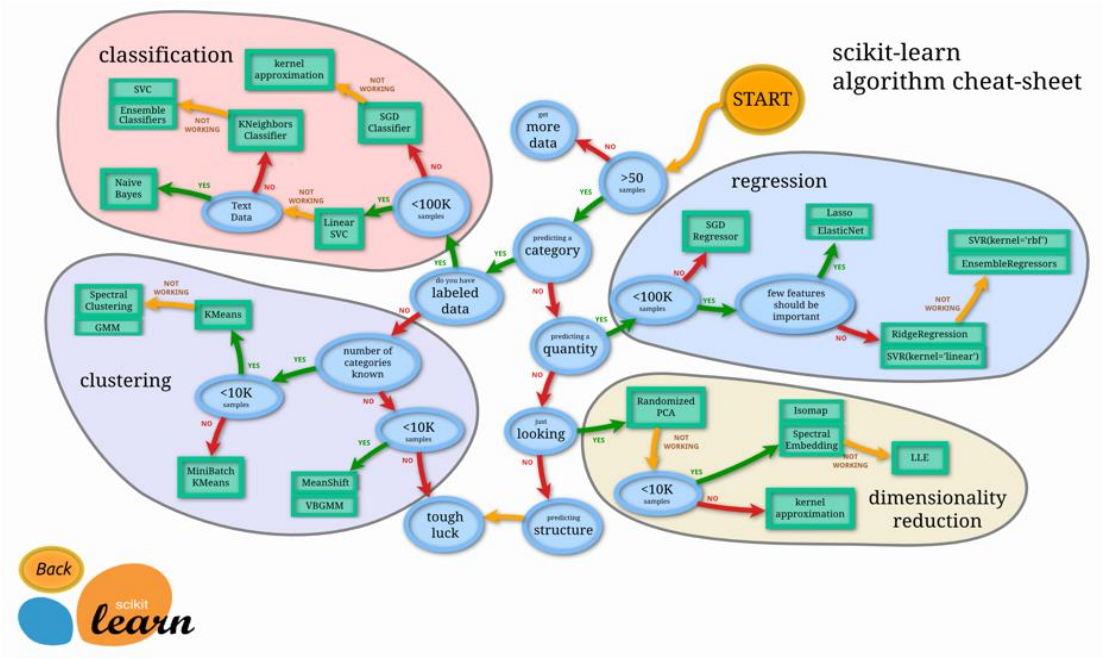


Fig. 3.3 Esquema para la elección del método de aprendizaje [23]

En el esquema de la Figura 3.3, podemos ver un esquema proporcionado por los creadores de la biblioteca scikit-learn que puede servir para la elección del método de aprendizaje que más se ajuste a nuestro problema. Si lo seguimos paso por paso podemos ver rápidamente que se trata para empezar de un problema de clasificación, ya que tenemos más de 50 datos, queremos predecir una categoría, y poseemos los datos etiquetados. Una vez dentro de la sección de clasificación, vemos que nuestra base de datos no es mayor de cien mil datos, y tampoco estamos utilizando texto como datos de entrada. Por lo tanto, los métodos que nos recomiendan son máquinas de vectores de soporte (SVC o Support Vector Classifier), K-Vecino más cercanos y clasificadores de conjuntos como son por ejemplo los bosques aleatorios. Aparte de estos 3 métodos de clasificación basados en el Machine Learning tradicional, también vamos a estudiar un método basado en las redes neuronales más simples.

3.2 Algoritmos de Clasificación de Machine Learning

En este apartado nos vamos a dedicar a dar una pequeña explicación de cada uno de los métodos que vamos a aplicar para la resolución de nuestro problema.

3.2.1 Arboles de decisión y bosques aleatorios

Los bosques aleatorios, es uno de los más potentes y utilizados para tareas de Machine Learning. Es tanto capaz de realizar tareas de regresión como de clasificación.

Este método es una evolución de los Arboles de Decisión, ya que consiste en crear un "bosque" con un determinado número de árboles de decisión. Por norma general, cuantos más arboles haya en el bosque, más precisa será nuestra predicción. Como acabamos de decir, este es un método que parte de los árboles de decisión, por lo que antes de adentrarnos en el funcionamiento de los bosques aleatorios, vamos a explicar en qué consisten los arboles de decisión.

Un árbol de decisión es un diagrama de flujo en el que cada nodo interno representa un test sobre cada uno de los atributos, cada rama representa el resultado del test y cada hoja, o nodo final, representa una clase, la cual será el resultado final.

La característica que hace que los árboles de decisión sean tan comúnmente utilizados, es que se pueden representar visualmente de forma que sean muy sencillos de entender. Otros de sus puntos positivos es que pueden trabajar tanto con datos numéricos, como con datos categóricos. En el caso de nuestra aplicación, una de las ventajas que más nos ha llamado la atención de este método es que no precisa que nuestros datos sean lineales para conseguir buenos resultados.

También encontramos una serie de desventajas en los árboles de decisión. Muchas veces, nuestro clasificador puede crear árboles demasiado complejos que no serán capaz de generalizar de forma correcta. Existen métodos para poder subsanar este error, los cuales son denominados como "poda" (pruning), de los cuales hablaremos más adelante. También pueden ser inestables, ya que un pequeño cambio en los datos puede

cambiar por completo nuestro árbol. En nuestro caso, la desventaja más importante, y por la cual se decidió descartar este método, para pasar a uno más evolucionado, es que los arboles de decisión crean lo que comúnmente se denomina algoritmos codiciosos (greedy algorithms), los cuales no nos garantizan que vayamos a tener una decisión óptima de manera global. Un algoritmo codicioso es aquel que crea una elección óptima de manera local, y espera que esta decisión también sea válida de forma global. Esto, como veremos más adelante, se puede solucionar creando múltiples árboles de decisión y no uno solo.

Se podría decir que los arboles de decisión trabajan muy bien con los datos con los cuales han sido entrenados, pero son muy poco flexibles cuando se trabaja con nuevos datos.

Ya que los árboles de decisión es un método sencillamente representable, vamos a ver un ejemplo para que se pueda entender de forma correcta. Aunque en nuestro caso, los datos del árbol van a ser numéricos, nos vamos a apoyar en un ejemplo que utiliza datos categóricos ya que así es más sencillo de comprender. Para este ejemplo vamos a utilizar un dataset que contiene información real de 887 de los pasajeros que viajaban en el Titánic [24]. Los datos que se tienen de estos pasajeros son, entre otros, edad, sexo, clase de pasaje (clase alta, media o baja), nombre, número de hijos a su cargo y como no, si fallecieron o no en la tragedia. Lo que se intenta con el arbole de decisión, es averiguar, si un pasajero con el que no hayamos hecho el entrenamiento sobrevivirá o no a la catástrofe. En el ejemplo, para que sea más sencillo, solo se han tenido en cuenta para la realización del árbol, el sexo, la edad, y la clase en la que viajaba el pasajero. En él, podemos ver como el algoritmo va haciendo preguntas sobre los atributos para finalizar obteniendo una clase.

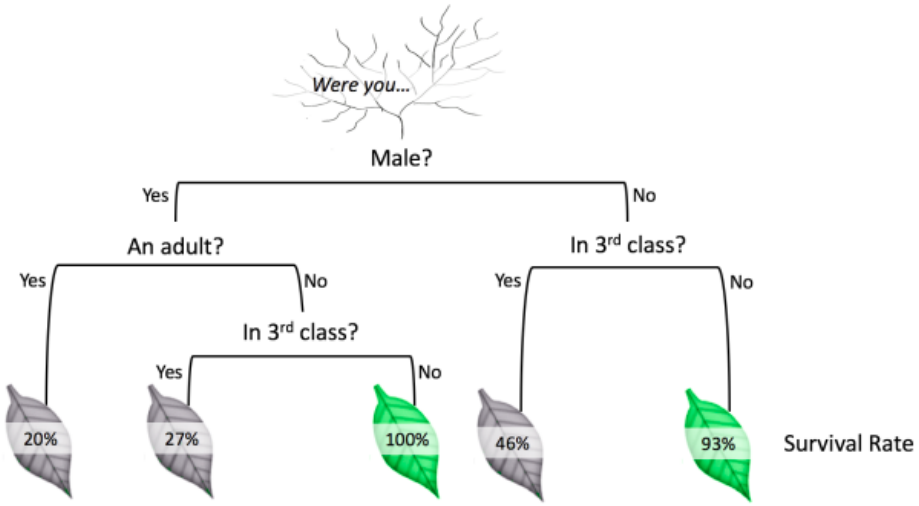


Fig. 3.4 Árbol de decisión Titánic [24]

Una vez hemos entendido el concepto de árbol de decisión, vamos a ver como se crea. La elaboración del árbol consiste, básicamente, en elegir que parámetros se van a utilizar, elegir las condiciones para separar los atributos, decidir cuándo parar y, por último, debemos "podar" el árbol.

Vamos a ver en primer lugar como decide el árbol la forma de hacer la separación. Cuando hablamos de separación nos referimos a la elección de cuando debemos ir por un lado de la rama o por la otra. Este punto es muy importante, ya que, si estas divisiones no están creadas de forma correcta, la precisión del algoritmo puede resultar bastante pobre. En la Figura 3.5 podemos ver un ejemplo grafico de cómo se realiza esta división. En ella podemos ver diferenciadas 2 clases, las cuales se han separado con las líneas a las cuales se les ha dado un valor numérico. En la Figura 3.6 podemos ver como el árbol de decisión, según si los atributos de la clase que estamos estudiando se encuentran a un lado o al otro de estas líneas, va a pertenecer a una clase o a otra.

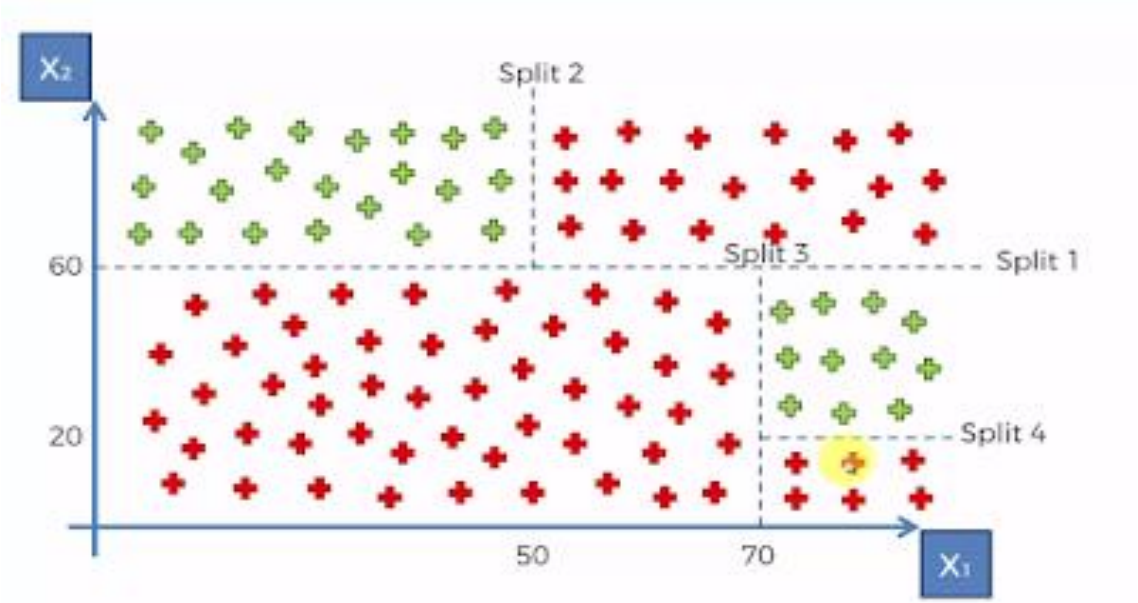


Fig. 3.5 Ejemplo de separación para cear un árbol de decisión [25]

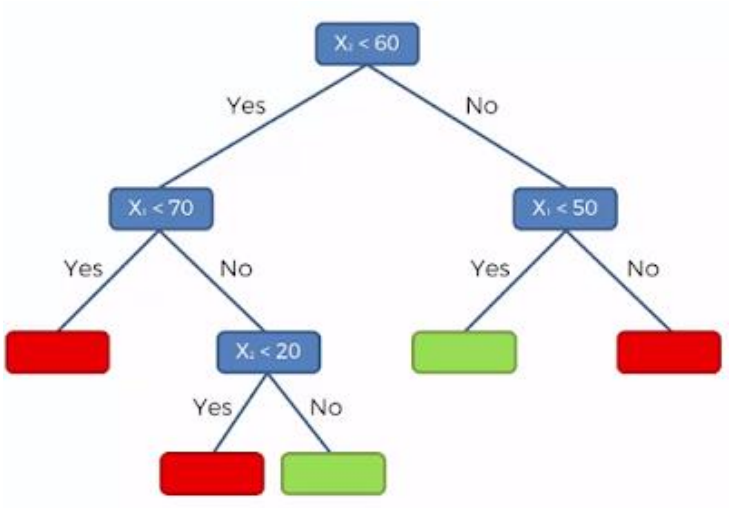


Fig. 3.6 Arbol de decisión simple [25]

Una vez visto esto, deberemos medir el grado de impureza de nuestras hojas, o clases. Cuando hablamos de impureza, hablamos de la certeza con la que nuestro clasificador va a seleccionar una clase. Imaginemos por ejemplo una clasificación binaria en la cual nuestro algoritmo tiene que decidir entre dos clases las cuales son sí o no. El resultado nunca va a ser cien por cien fiable, y el grado de fiabilidad, es lo que podemos llamar impureza. Existen diversos métodos para medir este coeficiente. Nosotros nos vamos a

centrar en los dos más populares, y que además vienen ya predefinidos en la biblioteca de Python de scikit-learn.

Coeficiente de Gini: Llamado así por el estadístico italiano Corrado Gini [26] el cual es su inventor. Este es uno de los métodos más sencillos de calcular, ya que simplemente aplica una pequeña formula estadística.

$$Gini = 1 - \sum_{i=1}^{c}(p_i)^2 \qquad \textit{Ecuación 3.1}$$

Siendo p la probabilidad de acierto de la clase, y c el número de clases. Esto da un coeficiente entre 0 y 1, el cual cuanto más próximo esté a cero, más puro significará que es nuestro resultado.

Entropía: Con entropía, al igual que en física, nos referimos al desorden que pueda tener nuestro clasificador.

$$Entropy = \sum_{i=1}^{c} - p_i \cdot log_2(p_i) \qquad \textit{Ecuación 3.2}$$

Cuanto menor sea el coeficiente de entropía, mejor será nuestro resultado. El rango de este coeficiente variará según el número de clases que tengamos y por ejemplo para dos clases la entropía máxima es 1 y para cuatro clases es 2, ya que la fórmula está en base-2.

En el capítulo 6 compararemos estas dos medidas sobre nuestro dataset para poder dar unas conclusiones sobre que método funciona mejor en nuestro clasificador.

Por último, solo quedaría podar nuestro árbol para así evitar el sobre entrenamiento. Como hemos dicho, uno de los problemas que nos podemos encontrar con los árboles de decisión es que podemos entrenarlos hasta que el número de ramas haga que la impureza sea cero, lo que hará que nuestro árbol trabaje bien con los datos de entrenamiento, pero que no sea capaz de trabajar de forma correcta con datos nuevos, al haberse convertido en un algoritmo demasiado complejo como para ser capaz de generalizar. En nuestro caso, el método que vamos a utilizar para ajustar los parámetros del árbol, para conseguir un árbol simple, que sea capaz de generalizar de forma

correcta, será el método de Grid Search explicado en el capítulo 5, y el cual analizaremos sus resultados más en detalle en el capítulo 6.

Una vez hemos explicado de forma general cómo funcionan los arboles de decisión, vamos a pasar a explicar los bosques aleatorios, y vamos a ver que mejoras presentan frente a este método anterior.

Los bosques aleatorios combinan la sencillez de los árboles de decisión con una mayor flexibilidad, lo que consigue un aumento enorme en la precisión de los resultados.

Como podemos intuir de su nombre, los bosques aleatorios son conjuntos de árboles de decisión. Para hacer un ejercicio de clasificación, cada árbol del bosque genera una etiqueta, para luego seleccionar la etiqueta que haya recibido más votos.

Para generar cada uno de estos estos árboles, el algoritmo lo que hace es crear un primer árbol, pero cogiendo solo algunos de los atributos que tenemos, los cuales han sido seleccionados aleatoriamente. Este proceso se realiza el número de veces que nosotros consideremos según el número de árboles que deseemos tener.

Este método tiene dos grandes ventajas frente a los arboles de decisión simples. La primera es que con los bosques aleatorios será mucho más sencillo evitar el sobre entrenamiento, aunque tengamos un alto número de árboles en el bosque. La otra razón para elegir este método es que trabaja de forma muy eficiente con bases de datos grandes y con varias dimensiones.

La desventaja que tiene este algoritmo es que no tenemos demasiado control sobre él. Podemos seleccionar el tipo de árbol que queremos, el número máximo de árboles o el número máximo de ramas, pero no tenemos más control estadístico sobre el modelo.

Como en todos los métodos, deberemos ajustar ciertos parámetros a la hora de entrenar nuestro clasificador para que trabaje de la mejor forma posible. En el capítulo de resultados analizaremos como afectan estos parámetros aplicándolos a nuestra propia base de datos:

- N_ESTIMATOR: Este parámetro sirve para seleccionar el número de árboles que queremos que tenga nuestro clasificador. Cuanto mayor sea el número de árboles que tengamos, más complejo será nuestro clasificador, por lo que puede no generalizar de forma correcta.
- CRITERION: Con este parámetro seleccionaremos que criterio de los vistos anteriormente queremos utilizar en nuestro clasificador para medir la impureza.
- MAX_DEPTH: Este último parámetro sirve para indicar el tamaño máximo que queremos que tenga cada uno de nuestros árboles. Al igual que el parámetro 'n_estimators', cuanto mayor sea este número, más complejidad tendrá nuestro árbol, y, por lo tanto, tendremos un mayor riesgo de sufrir sobredimensionamiento.

3.2.2 Máquinas de vectores de soporte

Más conocidas por su nombre en inglés, Support Vector Machines o SVMs, es uno de los modelos más utilizados en Machine Learning para tareas de clasificación y regresión de cualquier tipo de datos debido a que es una herramienta realmente poderosa, que ha mostrado poder llegar a un desempeño incluso mejor que el de otros tipos de métodos de aprendizaje supervisado como pueden ser las redes neuronales. La mayor ventaja de este método es su reducido coste computacional.

Este método consiste en un conjunto de algoritmos de aprendizaje supervisado que fue introducido por Vladimir Vapnik a principios de los 90 [27].

La idea inicial de este método es la de poder dibujar una línea recta que separe datos positivos y datos negativos dentro de un espacio bidimensional, pero el problema es que hay muchas líneas que se pueden dibujar y no sabemos cuál de ellas es la mejor elección. Según estipula este método, la mejor elección será aquella situada en el centro

del carril que separa los ejemplos negativos y los ejemplos positivos con mayor anchura. Podemos ver un ejemplo gráfico de esta línea en la Figura 3.7.

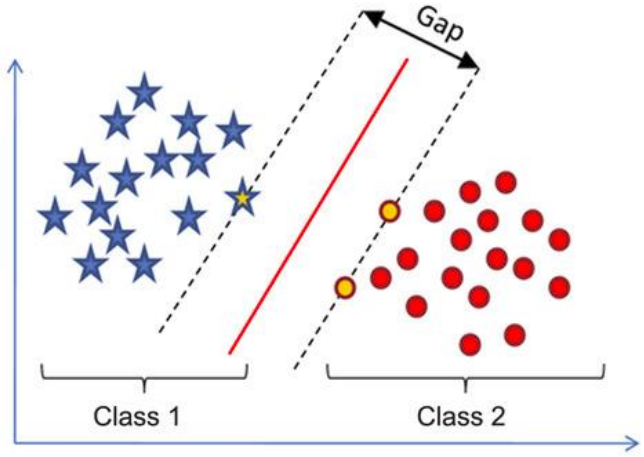


Fig. 3.7 Grafica Maquinas de vectores de soporte [28]

La obtención de esta línea se consigue de forma matemática, donde mediante el uso de vectores y diversos teoremas de álgebra y cálculo, se puede decidir si un punto se encontrará a un lado u a otro de dicha línea.

En muchas ocasiones los datos son demasiado complejos como para poder separarlos con una línea como en el ejemplo de la Figura 3.8. La solución a este problema es la de aplicar a nuestros datos las llamadas funciones kernel, para transformar el espacio bidimensional que teníamos en un principio, en un espacio con un determinado número de dimensiones en las cuales sí que seamos capaces de separar nuestros datos con una línea.

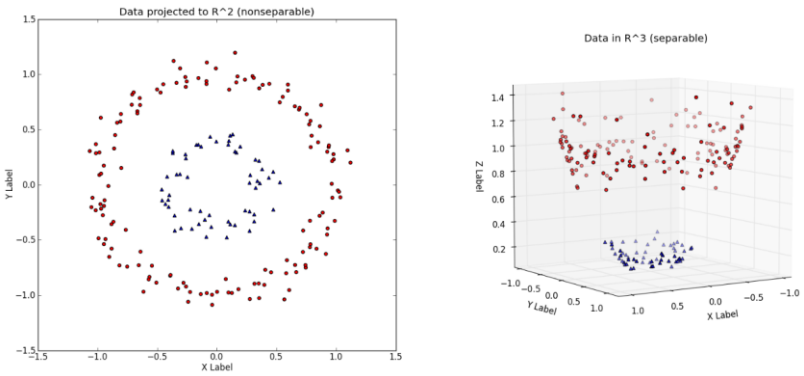


Fig. 3.8 Distribución de datos antes y después de aplicar el Kernel [29]

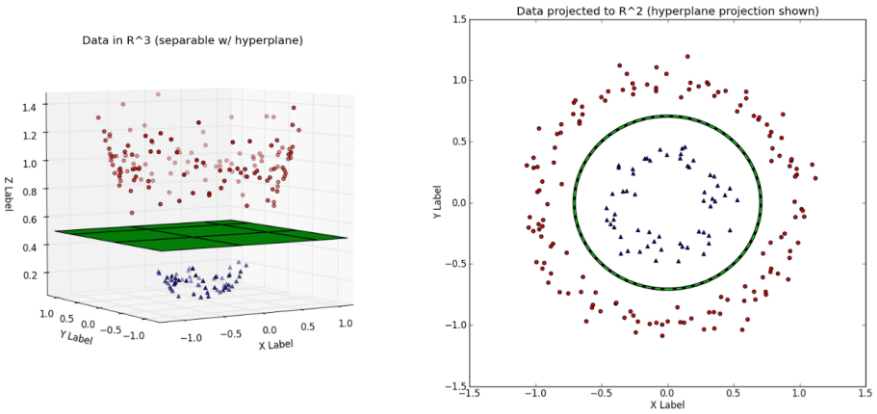


Fig. 3.9 Línea de separación con el kernel y tras deshacer la transformación [29]

Existen diferentes tipos de kernel o funciones que podemos aplicar para poder realizar esta separación como pueden ser lineal, polinómica o sigmoide. Elegir que kernel utilizar en problemas complejos como puede ser el nuestro puede ser realmente complicado, por lo que seleccionar nuestro kernel de forma teórica se puede considerar una tarea inviable. La forma que utilizaremos en nuestro kernel será la explicada en el capítulo 5 y aplicada en el capítulo 6, la cual denominamos Grid Search.

Al igual que en el resto de los métodos que estamos utilizando, a la hora de aplicarlo en nuestro programa de Python, deberemos dotar a nuestro clasificador de ciertos parámetros para que funcione de la forma más correcta posible. En el caso de las máquinas de vectores de soporte, existen un gran número de parámetros que se pueden ajustar para definir nuestro clasificador, pero nosotros nos basaremos en los más importantes y de mayor relevancia:

- C: El parámetro C, también llamado coste, sirve para seleccionar el valor o el peso que le damos a cada atributo a la hora de hacer la línea de separación. Cuanto mayor sea el peso que le demos a nuestros atributos, más restrictivo será nuestro clasificador. Un valor demasiado elevado del coste puede llevarnos a que nuestro clasificador no sea capaz de generalizar.
- KERNEL: El kernel ya lo hemos definido como la función que aplicamos a nuestros datos para poder hacer una separación lineal. En nuestro caso, los kernels que probaremos serán:

- ‘linear’: Se aplica una función lineal. Esta función intenta dibujar una línea recta para separar los datos en un plano bidimensional.
- ‘poly’: Se aplica una función polinómica de la siguiente forma, siendo ‘d’ el grado de la función.

$$k(x, y) = (x^T \cdot y + 1)^d \qquad \textit{Ecuación 3.3}$$

- ‘rbf’: Función gaussiana. Las siglas provienen de Radial Basis Function, traducido como función de base radial. Se aplica una función de la siguiente forma, siendo ‘γ’ la variable gamma la cual es uno de los parámetros que habrá que seleccionar en un futuro.

$$k(x, y) = e^{-\gamma|x-y|^2}, \gamma > 0 \qquad \textit{Ecuación 3.4}$$

- ‘sigmoid’: Esta kernel, también conocida como la de la tangente hiperbólica, proviene del campo de las redes neuronales en la cual se aplica la siguiente función, donde ‘γ’ y ‘c’ son variables las cuales seleccionaremos en un futuro.

$$k(x, y) = tanh(\gamma x^T y + c) \qquad \textit{Ecuación 3.5}$$

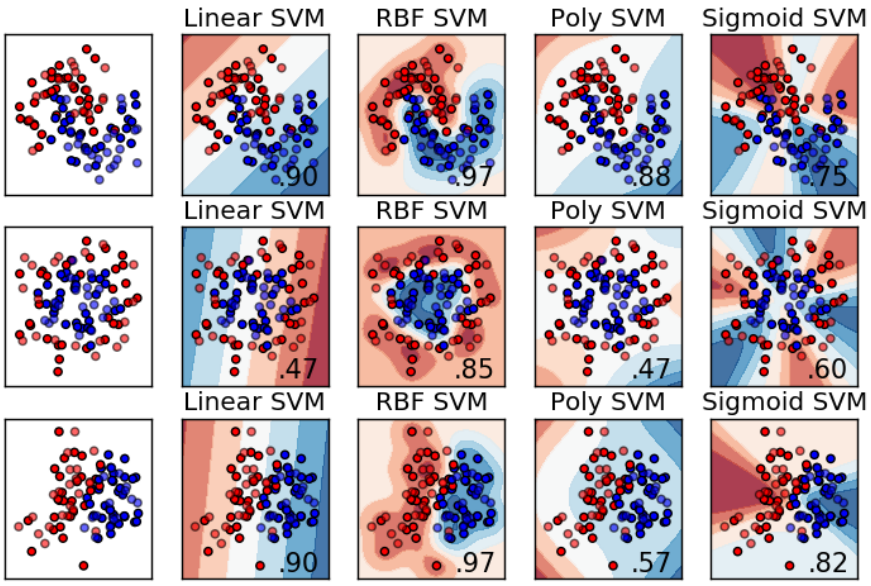


Fig. 3.10 Comparación gráfica de las distintas kernels [30]

En esta imagen podemos visualizar gráficamente un comparativo de las distintas kernels aplicadas a tres bases de datos diferentes.

En el caso de nuestra base de datos, realizaremos las pruebas únicamente con los kernel Linear y Poly ya que son las que mejor van a trabajar con nuestro tipo de datos.

- GAMMA: Gamma es el valor γ en las kernels no lineales, y define la importancia de cada ejemplo de entrenamiento.
- DEGREE: Grado de la función polinómica representado por 'd' en la ecuación 3.3.

3.2.3 K-Vecino más cercano

Más reconocido por su nombre en inglés K-Nearest Neighbour. Este método pertenece uno de los más utilizados por una sencilla razón. Es el más simple de todos y aun así es realmente potente. Al igual que muchos de los métodos estudiados en este proyecto, el algoritmo de k-vecino más cercano se puede utilizar tanto para tareas de clasificación como de regresión.

Este método es considerado no paramétrico. Esto es así porque para hacer una predicción, se requieren todos los datos de entrenamiento. Esto hace que cuantos más datos de entrenamiento tengamos, más complejo y grande será nuestro calificador. Por el contrario, los clasificadores que son considerados paramétricos, extraen ciertas características de los datos de entrenamiento, y una vez se han obtenido los parámetros para realizar la clasificación, ya no hacen falta más estos datos y pueden ser desechados. El mayor problema de que este método no sea paramétrico es que puede ser realmente lento en los casos en los que tengamos un dataset muy grande. Por esta misma razón, también se denomina a este método como un método vago ya que no construye ningún modelo a la hora de realizar el entrenamiento. Lo único que hace es guardar las instancias.

Lo que hace exactamente este algoritmo es almacenar todos los datos que le hemos introducido con sus respectivas clases, para que cuando le introduzcamos un nuevo dato, lo clasifique buscando con cuál de los datos anteriores tiene un mayor grado de similitud. La clase del nuevo valor será asignada según cual sea la clase más común

entre sus vecinos cuando está representada gráficamente. La variable K se utiliza para decirle a nuestro clasificador como queremos que sea de grande el radio de búsqueda a la hora de buscar el vecino más común, por lo que, si K equivale a 1, sólo nos fijaremos en el vecino más cercano. Veamos esto con un ejemplo gráfico.

Supongamos que los puntos azules pertenecen a una clase mientras que los puntos rojos pertenecen a otra distinta. Ahora añadimos un punto verde, el cual no hemos estudiado, por lo que no sabemos a qué clase pertenece. Lo que hará nuestro clasificador será dibujar un circulo alrededor de este nuevo punto para buscar que otros puntos están más cerca suya. El valor de K, el cual es un valor definido por el usuario, corresponderá con el número de ejemplos que estamos cogiendo alrededor de este nuevo punto. En la imagen de la Figura 3.11, podemos observar como este valor es muy importante y puede afectar completamente a nuestro resultado. Si nos fijamos en el círculo dibujado con línea continua, el valor de K sería 3, ya que nos estamos fijando en los tres vecinos más cercanos, entre los cuales tenemos dos puntos rojos y uno azul, por lo que el color más común será el rojo, y nuestro clasificador dará como solución que nuestro nuevo punto pertenece a la clase rojo. Por el contrario, si nos fijamos en la línea discontinúa, estaríamos utilizando un K igual a 5 donde nos encontraríamos con 2 puntos rojos y 3 puntos azules, por lo que nuestro clasificador supondrá que la nueva clase pertenece a los puntos azules. Con esto podemos ver que, cambiando ligeramente el valor de K, podemos obtener resultados totalmente distintos para un mismo dato.

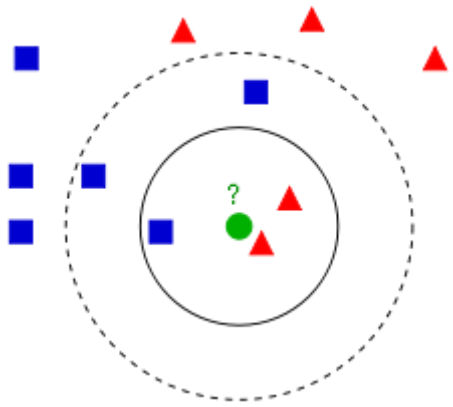


Fig. 3.11 Gráfico K-vecino más cercano [31]

El problema de esta constante K es que, si ponemos una K demasiado pequeña, podemos tener sobre entrenamiento y que el algoritmo no sea capaz de generalizar de forma correcta. En la Figura 3.12, podemos ver dos ejemplos de cómo se ha trazado la línea de separación según la K que tengamos. En el ejemplo de la derecha, donde tenemos que K es igual a 1, hay sobreajuste (overfiting), ya que si introducimos un nuevo valor solo nos fijaremos en el punto más cercano, y en muchos casos no será el correcto. Si el test del clasificador se hiciese con los mismos datos que tenemos en el entrenamiento, obtendríamos un resultado del 100 por cien, pero con nuevos datos fallaría bastante. En cambio, en el ejemplo de la izquierda, por mucho que haya algunos puntos rojos en la zona azul y viceversa, nuestro clasificador está haciendo un ejercicio correcto de generalización, el cuál será lo que necesitemos a la hora de introducir nuevos datos.

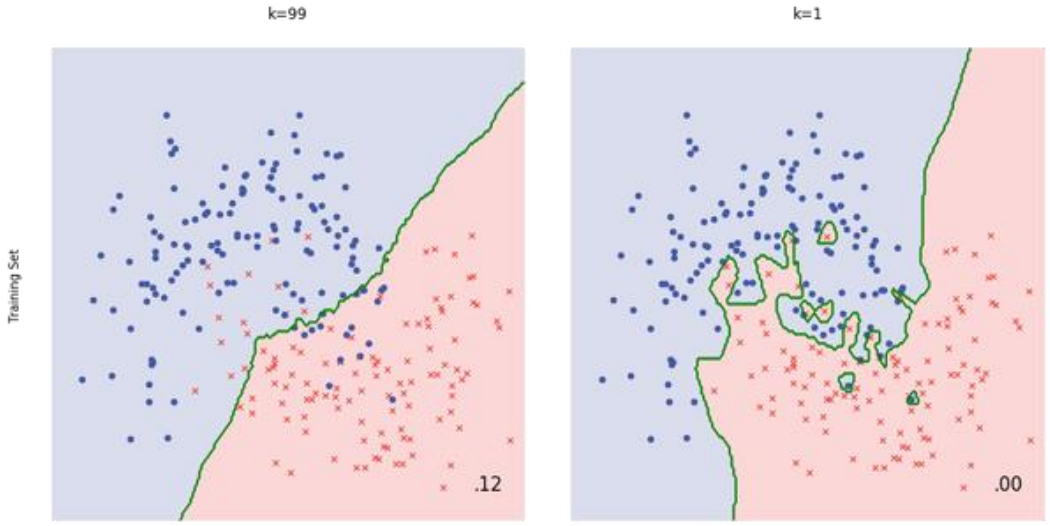


Fig. 3.12 Ejemplo de separación con y sin sobre entrenamiento dependiendo del valor de K [32]

El cálculo teórico de la K en casos como el nuestro, en el cual nuestros datos tienen una distribución altamente compleja y no lineal, es un poco arbitrario. Nosotros basaremos la elección de la constante al método de Grid Search, el cual hemos definido en el capítulo 5, y veremos los resultados aplicados a este algoritmo en el capítulo 6.

3.2.4 Perceptrón Multicapa

El método del perceptrón multicapa es uno de los métodos más simples basado en redes neuronales que nos podemos encontrar. La principal ventaja de este método frente al resto de algoritmos basados en redes neuronales es que no tiene un coste computacional tan alto, por lo que podemos realizar aplicaciones que funcionen de forma fluida en tiempo real sin requerir el uso de un ordenador de alto rendimiento. Este método se utiliza sobre todo para sistemas de reconocimiento de imágenes y reconocimiento de voz.

Como su nombre indica, una red neuronal es un sistema basado en el comportamiento de las neuronas del cerebro humano. En nuestro caso, definiremos una neurona como algo que tiene un número alejado dentro. Una red neuronal, por lo tanto, será un conjunto de neuronas conexionadas entre sí para formar una red. Estas neuronas están distribuidas por capas. Tendremos una capa de entrada, la cual contendrá la información acerca de nuestra imagen. En nuestro caso particular, cada una de nuestras imágenes está compuesta por 784 pixeles con valores que van desde el 0 al 255, correspondiéndose el cero con el negro, el 255 con el blanco y el resto de los valores intermedios se corresponderán con su escala de gris equivalente, como veremos en el próximo capítulo. Es por esto por lo que nuestra capa de entrada estará compuesta por 784 neuronas, cada una con un valor de entre 0 y 255, correspondiéndose con los pixeles de las imágenes. Tendremos también una capa de salida. En nuestra aplicación, por ejemplo, esta capada estará compuesta por 5 neuronas, numeradas del uno al cinco, las cuales se corresponderán con las clases que tenemos disponibles. Entre medias de estas dos capas principales, tendremos las que se denominadas capas ocultas. La elección de la cantidad de estas capas y su tamaño, para tener un algoritmo que funcione de forma correcta, es bastante arbitrario, y se precisa de hacer diferentes pruebas para poder seleccionar los mejores valores. Nosotros seleccionaremos estos valores apoyándonos del método de Grid-Search.

En el ejemplo de la Figura 3.13 tendremos una capa de entrada con 8 nodos o datos de entrada, seguidas de 3 capas intermedias las cuales están compuestas por 9 nodos cada una, y finalmente una capa de salida dispuesta por 4 nodos o posibles salidas.

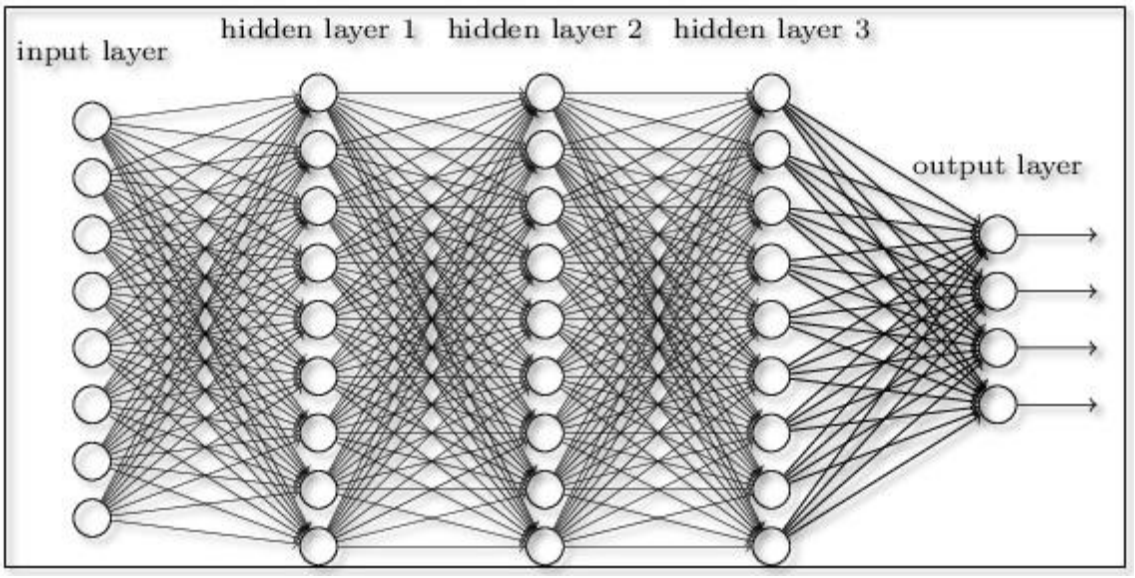


Fig. 3.13 Ejemplo de red neuronal [33]

Las neuronas de las capas intermedias se activan según la información de las neuronas predecesoras. Estas neuronas, tendrán información de los distintos patrones que pueda tener la silueta de una mano en la imagen, y dependiendo de si estos patrones se encuentran en la imagen de entrada, la neurona recibirá un peso distinto, el cual hará que se active o no dicha neurona. Existen diferentes funciones de activación los cuales nos dirán si la neurona se deberá activar en función de los pesos que haya recibido.

La elección de la función de activación que vayamos a utilizar, la realizaremos utilizando el método de Grid-Search explicado en el capítulo 5. Las funciones que probaremos serán la función sigmoide, la función tangente hiperbólica y la función linear rectificada unitaria. Esta función se añadiría a la función básica de las redes neuronales la cual es la siguiente:

$$(x_1 w_1 + x_2 w_2 + \cdots + x_n w_n - bias) \qquad \textit{Ecuación 3.6}$$

En la cual, las ‘x’ se corresponden con los atributos de la imagen, las ‘w’ se corresponden con el peso que se le da a cada atributo y la constante ‘bias’ será un valor que se asignará a cada neurona para decidir si se debería activar o no.

Lo último que nos faltaría por analizar es la optimización de los pesos asignados para poder minimizar el error. Los algoritmos más utilizados para obtener este error mínimo son L-BFGS y el gradiente descendiente estocástico. En nuestro caso utilizaremos el algoritmo L-BFGS, ya que es el que mejor trabaja cuando tenemos bases de datos pequeñas, como la nuestra. Las siglas de este método provienen de "Low memory - Broyden – Fletcher – Goldfarb - Shanno" debido a sus creadores. Este método es considerado quasi-Newton ya que permite conseguir el mínimo de una función sin necesidad de su matriz Hessiana.

Para finalizar con la teoría acerca del perceptrón multicapa nos falta definir los parámetros que analizaremos en el capítulo de resultados.

- "HIDDEN_LAYER_SIZES": Este parámetro sirve para ajustar el tamaño que queremos que tengan las redes ocultas de nuestra red neuronal.
- "ACTIVATION": Con este parámetro seleccionaremos que función de activación queremos utilizar de entre las siguientes:

    - "relu": Se corresponde con la función linear rectificada unitaria.

$$f(x) = max(0, x) \qquad \textit{Ecuación 3.7}$$

    - "logistic": Se corresponde con la función sigmoide.

$$f(x) = \frac{1}{(1+ e^{-x})} \qquad \textit{Ecuación 3.8}$$

    - "tanh": Se corresponde con la función hiperbólica.

$$f(x) = tanh(x) \qquad \textit{Ecuación 3.9}$$

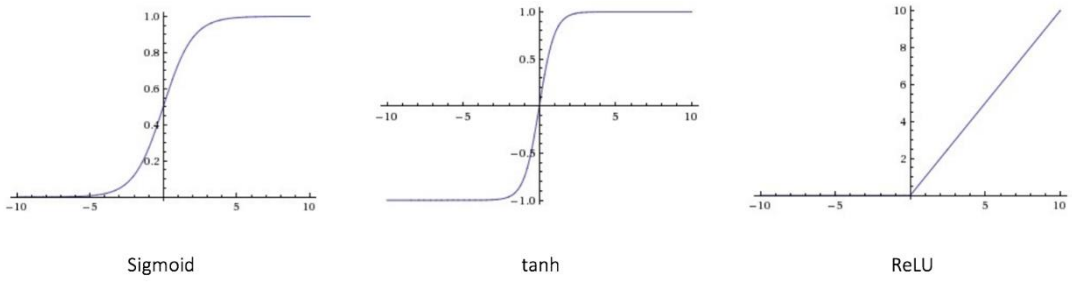


Fig. 3.14 Funciones de activación [34]

En la Figura 3.14 podemos ver representadas las 3 funciones de activación enunciadas.

Una vez vistos los distintos métodos de Machine Learning con los que vamos a trabajar, pasemos ahora a entender en que consiste la visión artificial.

4 VISION ARTIFICIAL

La visión artificial consiste en transformar imágenes del mundo real, en algo que sea capaz de interpretar un ordenador, para así poder trabajar con ellas.

El concepto de visión artificial surge en los años 60, con la intención de ser capaces de conectar una cámara de video a un ordenador, y no solo ver o representar las imágenes captadas por la cámara, sino ser capaces de interpretar lo que estamos viendo en esas imágenes. El primer algoritmo que aplicaba la visión artificial fue desarrollado por Larry Roberts [35], el cual consistía en contemplar una serie de bloques dispuestos en una mesa, y que el ordenador fuese capaz de representar estos bloques desde otra perspectiva distinta.

Podemos nombrar ciertos trabajos que nos han resultado interesantes dentro del campo de la visión artificial. En primer lugar, vamos a mencionar trabajos que han utilizado cámaras de luz estructurada similares a la utilizada en nuestro proyecto. En el trabajo realizado por Park, Kim, Na, Yi y Turk [36], utilizan una cámara de luz estructurada para poder obtener el contorno de ciertos objetos, entre ellos la mano. Otro trabajo interesante acerca de las cámaras de luz estructurada es el realizado por DePiero y Trivedi [37], en el que explican la teoría de este tipo de cámaras y dan varios ejemplos de aplicación. Finalmente, artículos como el realizado por Araujo [38] o por Chen, Kim Liang, Zhang y Yuan [39], han sido realmente útiles a la hora de filtrar nuestra imagen y obtener una región de interés adecuada para la realización de nuestro proyecto.

Para poder entender la visión artificial de forma que podamos trabajar con ella, primero deberemos entender cómo interpreta un ordenador cada imagen que le mostramos. Para un ordenador, una imagen no es más que una matriz compuesta por una serie de números. En la Figura 4.1 podemos ver la forma en la que el ser humano ve una imagen, y la forma en la que un ordenador ve esa misma imagen.

$$\rightarrow \begin{bmatrix} 87 & 138 & 191 & 160 & 228 & 197 & 6 \\ 133 & 121 & 106 & 207 & 85 & 75 & 5 \\ 30 & 236 & 27 & 149 & 161 & 218 & 95 \\ 157 & 141 & 17 & 101 & 187 & 23 & 42 \\ 243 & 78 & 122 & 172 & 151 & 51 & 104 \\ 30 & 42 & 8 & 25 & 13 & 209 & 128 \\ 114 & 7 & 206 & 79 & 84 & 90 & 118 \\ 251 & 49 & 250 & 110 & 240 & 188 & 99 \\ 190 & 109 & 139 & 12 & 184 & 251 & 36 \end{bmatrix}$$

Fig. 4.1 Hombre y mujer caminando y su matriz equivalente

De la imagen de la izquierda podemos realizar un análisis exhaustivo a simple vista, sin necesidad de pensar demasiado. Podemos ver a dos personas de las cuales una es un hombre y otra es una mujer. Conseguimos definir cómo van vestidos, diferenciar 3 zonas, el fondo, el suelo o carretera y a las dos personas en cuestión. Un ordenador, sin embargo, no es capaz de ver nada de lo que nosotros estamos viendo a simple vista. La máquina lo único que vera será algo parecido a la matriz de la imagen 4.1, y nosotros deberemos entrenarle, para que sea capaz de identificar los diferentes objetos en esta matriz. El ordenador, por lo tanto, lo que vera será una o varias matrices, compuestas por números comprendidos del 0 al 255 los cuales se corresponderán con cada pixel de la imagen. Dependiendo del espacio de color con el que estemos trabajando, el ordenador tendrá que interpretar estos valores de una forma o de otra.

Veamos que es un espacio de color, y los distintos espacios que existen y con los que podemos trabajar.

4.1 Espacio de color

Primero empecemos explicando que entendemos por color. Para el ser humano, el color es la impresión que producen en la retina los rayos de luz reflejados y absorbidos por un cuerpo, según la longitud de onda de estos rayos.

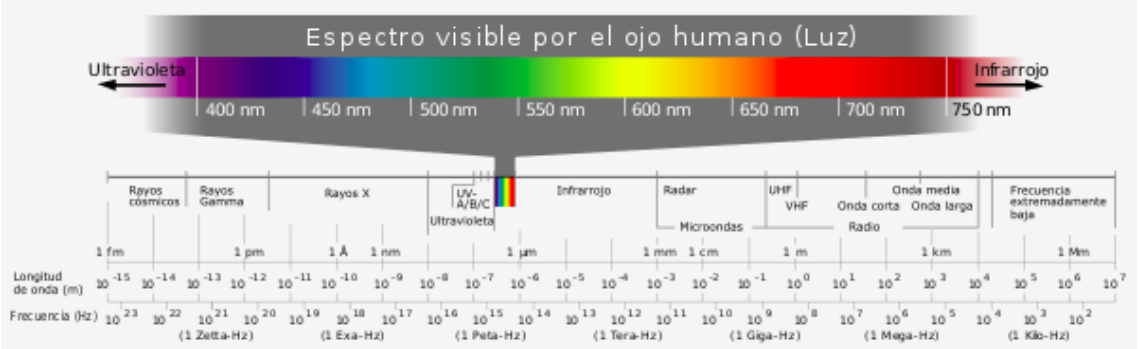


Fig. 4.2 Espectro visible por el ojo humano

En la imagen de la Figura 4.2, vemos cual es el espectro visible por el ojo humano, el cual corresponde con los valores comprendidos entre los 400nm y los 750nm de longitud de onda aproximadamente. Todo lo que queda fuera de este rango, es imperceptible de forma natural para el ser humano. No obstante, no todos los seres vivos tienen en mismo espectro visible, por ejemplo, muchos animales como los ratones, son capaces de percibir la radiación ultravioleta a través de la vista.

Una vez definido lo que es el color, vamos a explicar que son los espacios de color, y cuáles de ellos son los más comunes a la hora de trabajar en visión por computador o inteligencia artificial.

Un espacio de color es una representación matemática de un rango de colores. Trabajar con diferentes espacios de color puede traer sus ventajas y sus inconvenientes. Existen infinidad de espacios de color, como por ejemplo RYB (Red, Yellow, Blue), utilizado para conceptos de arte y pintura, CMYK (Cyan, Magenta, Yellow, Key-black) utilizado por la mayoría de impresoras, YIQ (Iluminance, In-phase, Cuadrature) utilizado principalmente en televisores americanos y japoneses, y un largo etcétera.

Nosotros nos vamos a centrar en los tres espacios de color más utilizados para tareas de visión por computador, los cuales son RGB, HSV y escala de grises.

4.1.1 RGB

Este es el espacio de color más común de todos, ya que es el que más se asemeja a la visión del ser humano. Las siglas RGB vienen de Red, Green y Yellow, que son los colores que utiliza este espacio de color para formar cualquier otro color dentro del espectro de visión del ser humano. La gran mayoría de los sistemas de imagen digital usan estos 3 canales para reproducir un color. Lo bueno de este espacio de color, es que cuando vemos una imagen definida por estas características, vemos los colores como los veríamos en el mundo real. Como hemos dicho antes, un ordenador ve las imágenes como matrices compuestas por números comprendidos entre el 0 y el 255. En el caso de las imágenes en formato RGB, el ordenador lo que verá serán tres matrices de números comprendidos como siempre entre el 0 y el 255. Cada matriz se corresponderá con uno de los componentes del espacio RGB, por lo que tendremos una matriz para el rojo, otra matriz para el verde, y una última matriz para el azul. La suma de estas tres matrices es lo que nos dará una imagen comprensible para el ojo humano.

Por poner un ejemplo, vamos a ver una composición de tres matrices de 2 x 2, lo que equivaldría a una imagen de 2 x 2 pixeles (4 pixeles).

$$\begin{bmatrix} 255 & 0 \\ 255 & 0 \end{bmatrix} + \begin{bmatrix} 255 & 255 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 255 \end{bmatrix} =$$

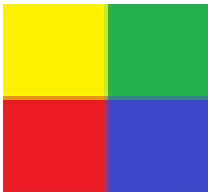


Fig. 4.3 Suma de matrices RGB convertidas en imagen

El RGB puede resultar útil cuando trabajamos con colores llamativos, o poco comunes en un entorno estándar. Si el color del objeto que estamos buscando, es de un tono llamativo, o de un color que no se encuentre en más objetos en el entorno que lo rodea, será relativamente sencillo filtrar la imagen para ver solo el color deseado. Si, por ejemplo, estamos trabajando con un color más neutro, como puede ser el de la piel humana, será mucho más complicado llevar a cabo este filtrado, ya que, en el caso de querer centrarnos en las manos, el filtrado no eliminará nuestra cara.

Otro de los inconvenientes de este espacio, es que es muy sensible a la luz. Un objeto puede tener unos valores de RGB muy distintos según lo iluminado que este.

Por último, éste espacio de color, así como de todos los espacios de color que requieren 3 matrices para definirse, tienen un alto coste computacional. Si, por ejemplo, tenemos una imagen de tamaño 64*64 pixeles, estaremos trabajando finalmente con 12288 datos. En el caso de que nuestro dataset sea extenso, esto puede afectar a que los tiempos de procesado demoren demasiado.

4.1.2 HSV

Este método creado en 1978 por Alvy Ray Smith [40], cofundador de Pixar, es una representación alternativa al espacio RGB. Las siglas HSV provienen de Matiz (Hue), Saturación (Saturation) y Valor (Value). El matiz es el valor que nos da la información sobre el color, la saturación es la cantidad de ese color que hay presente, y el valor se corresponde con la cantidad de brillo, o cantidad de blanco, que hay en ese determinado color.

En este caso también tendremos matrices con valores comprendidos entre el 0 y el 255 para cada variable. Como podemos ver, al igual que en el espacio RGB, estaremos trabajando con 3 matrices por cada imagen, por lo que seguimos teniendo el problema que teníamos de procesado.

La razón por la cual este espacio es uno de los más utilizados es porque solo uno de sus valores depende de la luz, por lo que, con un pequeño ajuste al principio de la ejecución, seremos capaces de obtener buenos resultados.

La forma más utilizada para representar este espacio de color para así entenderlo, y a su vez compararlo con el espacio RGB es con un cono en el cual se ven representados los 3 valores H, S y V.

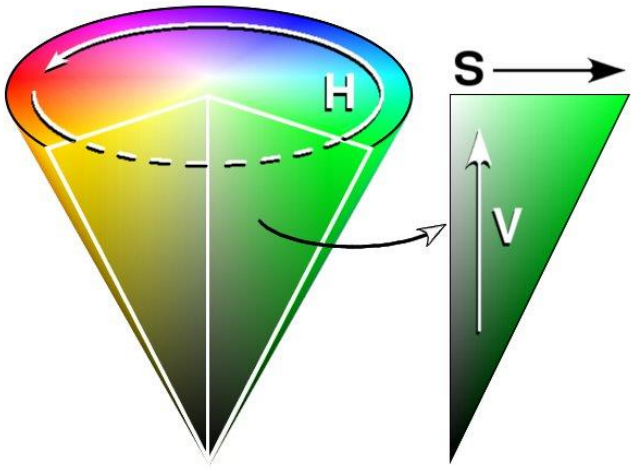


Fig. 4.4 Cono de transformación RGB-HSV

En nuestro caso no hemos tenido que utilizar este espacio de color, ya que hemos utilizado una cámara infrarroja para la captación de imágenes. Aun así, creo que es importante definir este espacio de color, ya que en el caso de no tener a disposición una cámara de las características de la utilizada, este sería el mejor espacio con el que trabajar en nuestra aplicación.

4.1.3 Escala de Grises

Este es uno de los espacios más sencillos de todos, ya que en este caso solo vamos a utilizar un canal. Por lo tanto, solo estaremos trabajando con una matriz por cada imagen. Es por esto por lo que aquí el problema computacional que teníamos con el resto de los espacios se ve altamente reducido, exactamente, dos terceras partes.

Una variante muy utilizada de la escala de grises, son las imágenes binarias. Este tipo de imágenes se basan en una matriz binaria en la cual solo tenemos dos valores. El cero corresponderá con el blanco y 1 con el negro. es el más simple de todos, y por lo tanto el que tiene los tiempos de procesado más rápidos.

En la mayoría de los casos cuando se trabaja con imágenes, primero se realiza el filtrado de la imagen en uno de los espacios de color tridimensionales que hemos visto

anteriormente, para luego convertirlos en escala de grises a la hora del entrenamiento y el procesado de los algoritmos de inteligencia artificial.

Fig. 4.5 Comparación de imagen en RGB, HSV y escala de grises

En la Figura 4.5 podemos ver aplicados los 3 espacios de color definidos sobre una misma imagen.

4.2 Visión 3D

En nuestro caso hemos aprovechado la disponibilidad de una cámara de profundidad para la realización del proyecto, lo cual ha facilitado enormemente nuestro trabajo a la hora de captar y filtrar la imagen de la mano.

Existen diferentes métodos para capturar imágenes y poder captar su profundidad en el espacio. En nuestro caso, el método que utiliza la cámara con la que se han tomado las imágenes es el de la luz estructurada, la cual funciona de la siguiente forma.

Para llevar a cabo este método hacen falta, como mínimo, un proyector y un sistema de adquisición, el cual será comúnmente una cámara. El proyector genera un patrón de luz conocido, el cual se deforma al incidir sobre el objeto que estamos utilizando. La cámara capta estas deformidades de la proyección con la cuál realiza los cálculos necesarios para darnos una imagen con los que podamos trabajar.

La luz utilizada por el proyector es una luz infrarroja. La razón de utilizar una luz que está fuera del rango visible es para que no interfiera con la captación de imágenes de otro tipo de cámaras, como por ejemplo una cámara RGB.

Nuestra cámara tiene un alcance de entre 20 y 120cm, lo cual la hace perfecta ya que es el rango de distancias con los que vamos a trabajar, con lo cual será muy sencillo ya que, si colocamos la mano en ese rango de distancias de la cámara, podremos filtrar el fondo con mucha facilidad y ver solo la imagen de la mano.

La mayor ventaja de utilizar una cámara de luz infrarroja es que tanto la luz ambiental como el color de la piel, no afectaran en absoluto a la toma de imágenes, por lo que no habrá que realizar ningún ajuste previo antes de su uso.

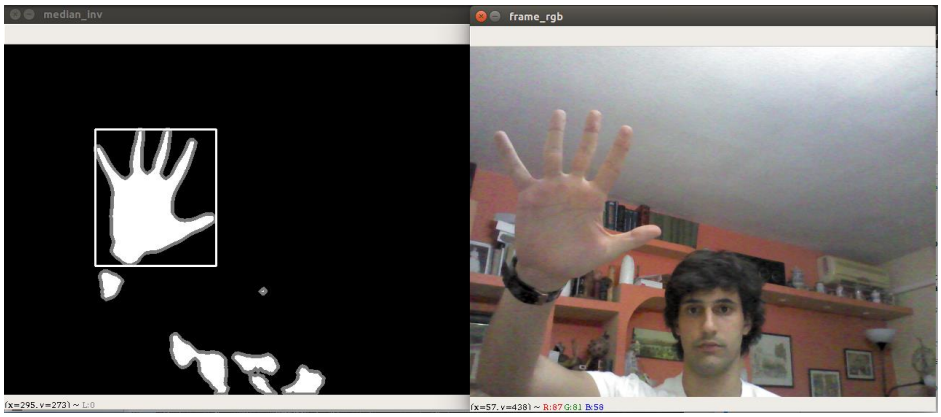


Fig. 4.6 A la izquierda imagen infrarroja filtrada. A la derecha imagen captada por la cámara RGB en condiciones óptimas de luz

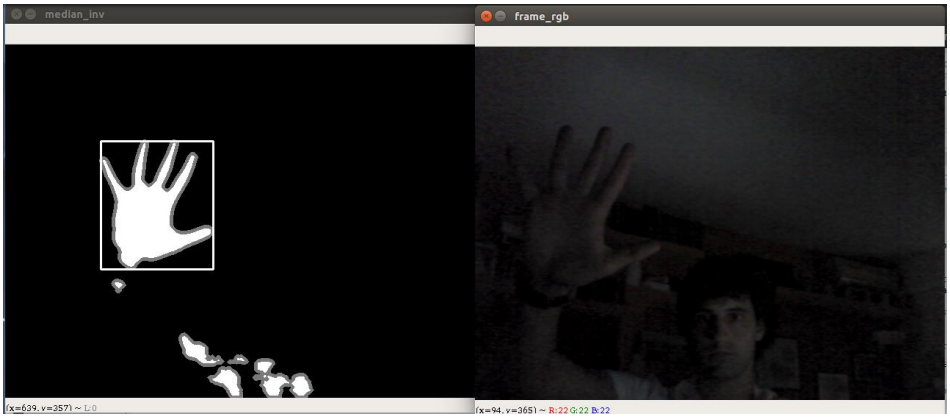


Fig. 4.7 A la izquierda imagen infrarroja filtrada. A la derecha imagen captada por la cámara RGB en condiciones de poca luz

En las Figuras 4.6 y 4.7, podemos apreciar como tomando imágenes prácticamente iguales, una con una iluminación correcta, y la otra prácticamente a oscuras, el resultado de la obtención con la imagen infrarroja es exactamente el mismo.

En el siguiente capítulo pasaremos a explicar de forma ordenada como ha sido el proceso para la creación del algoritmo para el reconocimiento de los dedos de la mano

5 PROCESO

En este apartado vamos a explicar cómo es la estructura que tiene un proceso completo de Machine Learning, el cual incluye en primer lugar la obtención del dataset, la reestructuración de los datos del dataset para poder aprovecharlos en el algoritmo, la distribución entre datos de entrenamiento y datos de test, la declaración del nuestra herramienta de clasificación así como de los parámetros que van a determinar su funcionamiento, el entrenamiento en sí, la fase de predicción y finalmente la obtención y análisis de los resultados.

En la Figura 5.1 podemos ver un diagrama de flujo del orden en el que hemos realizado cada acción dentro de nuestro algoritmo y de cómo se debería realizar cualquier algoritmo de Machine Learning.

En primer lugar, deberemos tener una buena base de datos. En caso de no tenerla, de no tenerla, deberemos o bien, buscarla por internet, o hacerla nosotros mismos. Una vez las tengamos bien organizadas, deberemos cargarlas a nuestro programa para poder trabajar con ellas. Habrá que declarar los clasificadores que vayamos a utilizar, y seleccionar las distintas variables o parámetros que ajustan los clasificadores para que trabajen de la mejor forma posible. Cuando tengamos seleccionados unos parámetros que consideremos que pueden hacer que el clasificador funcione de forma correcta, realizaremos el entrenamiento y haremos una serie de pruebas. En este caso las pruebas se harán cogiendo parte de las imágenes de la base de datos para el entrenamiento, y parte para las predicciones. Si los resultados no son lo suficiente buenos deberemos repasar los parámetros que hemos seleccionado para volver a realizar el entrenamiento. En el caso de tener porcentajes altos de acierto, podremos pasar a la ejecución en tiempo real, para ver cómo se comportan en este caso.

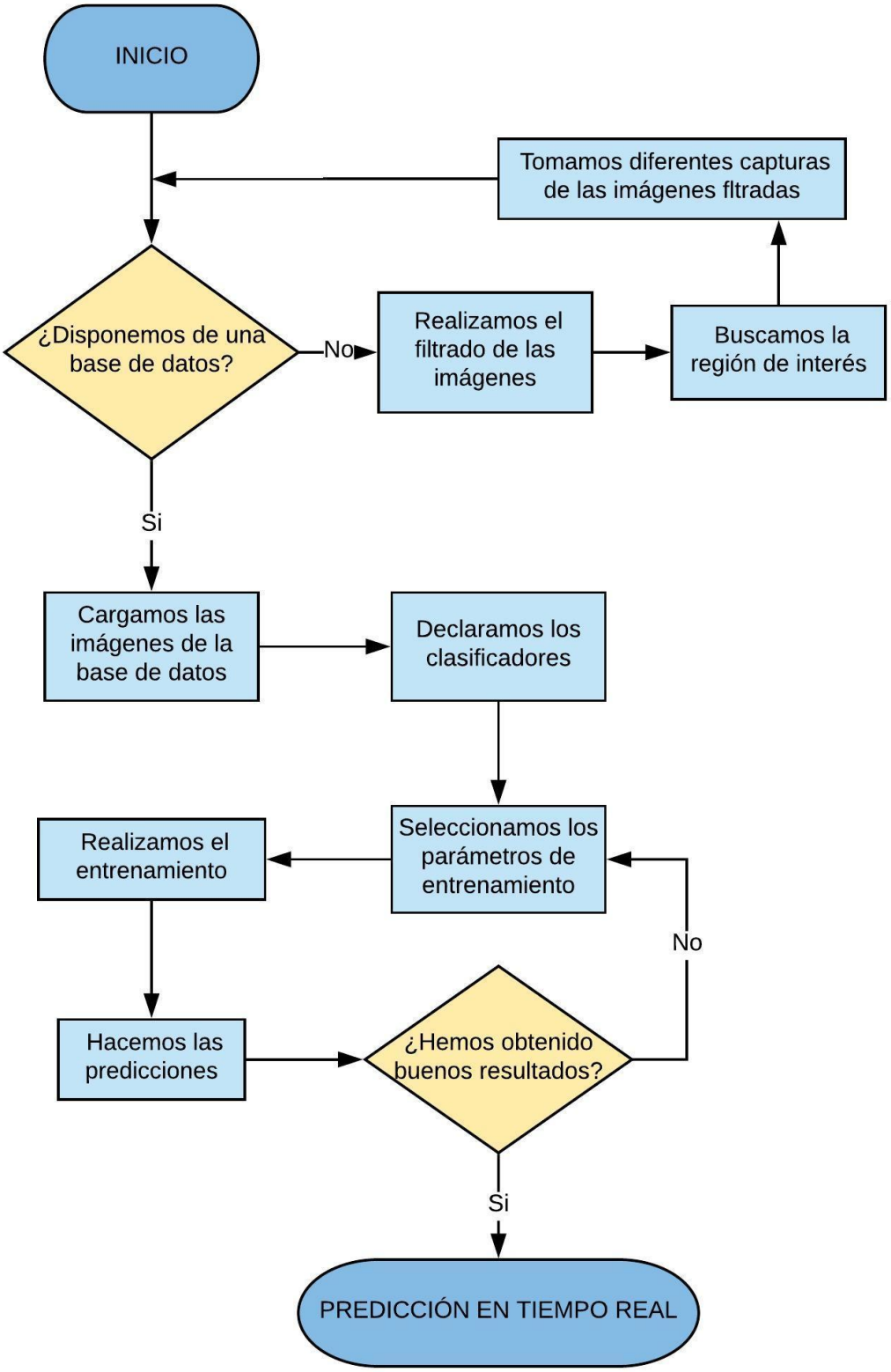


Fig. 5.1 Diagrama de flujo del proceso

5.1 Filtrado y obtención de las imágenes

En primer lugar, vamos a detallar el proceso que se ha llevado a cabo para el filtrado de las imágenes obtenidas por la cámara infrarroja. Este punto es de vital importancia, ya que un buen filtrado de las imágenes nos facilita muchísimo a la hora de la elaboración de una base de datos. Este punto también será esencial en la ejecución en tiempo real, ya que, para tener una correcta predicción, las imágenes que introduzcamos al clasificador deberán ser del mismo tipo que las cuales con las que le hemos entrenado.

En la Figura 5.2, podemos diferenciar los dos tipos de imágenes de entrada que recibimos a través de la cámara utilizada. La imagen de la izquierda se corresponderá con la imagen captada por la cámara RGB, y la imagen de la derecha, con la captada a través de la cámara infrarroja. En esta Figura podemos ver un ejemplo de cómo situando la mano en el rango que viene en las especificaciones de la cámara, automáticamente, todo lo que quede fuera de ese rango se verá en una tonalidad de verde homogénea, mientras que en la imagen de la mano veremos las líneas generadas por el proyector de luz infrarroja.

De aquí en adelante trabajaremos exclusivamente con la imagen recogida por la cámara de luz infrarroja. La cámara RGB nos sera util solo para poder interpretar de forma mas sencilla lo que esta ocurriendo.

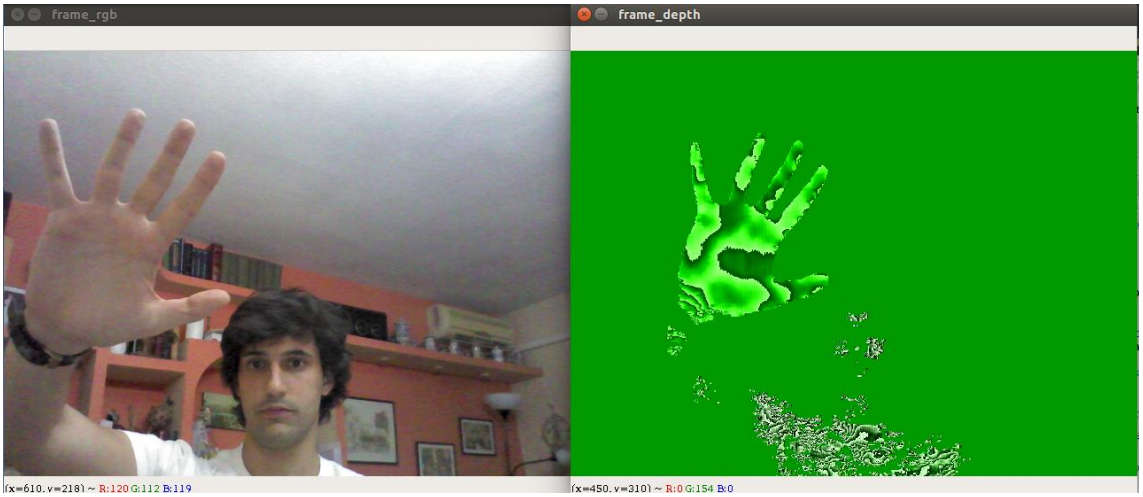


Fig. 5.2 A la izquierda imagen RGB. A la derecha imagen infrarroja sin tratar

Una vez tenemos esta imagen, deberemos aplicar una serie de filtros para poder trabajar con mayor comodidad.

En primer lugar, debemos saber que, aunque la imagen haya sido capturada a través del método de luz infrarroja proyectada, el software de captación lo que nos da es una imagen RGB como la que podemos ver en la parte derecha de la Figura 5.2.

Aplicando una simple máscara podemos eliminar toda la parte de la imagen que corresponde al fondo, y al mismo tiempo binarizar la imagen. Lo que haremos será aplicar lo que se denomina como mascará. Con esta mascara simplemente convertiremos todos los pixeles que incluyan el color que queremos eliminar en blanco, y el resto de los pixeles en negro. Determinar cuál es exactamente el color de los pixeles que queremos eliminar es relativamente sencillo, ya que en la ventana emergente que creamos para mostrar la imagen capturada por la cámara infrarroja, podemos ver en la esquina inferior izquierda los valores RGB del pixel que se encuentra situado debajo del puntero. En la imagen de la de derecha de la Figura 5.2, podemos ver que nos marca los valores R:0 G:154 B:0, que es el valor con el que tendremos que aplicar nuestra mascara. En el caso de tener una imagen en escala de grises, solo nos aparecería un valor, el cual se corresponderia con el valor de gris que hay en el pixel debajo del puntero, como podemos ver en la imagen de la derecha de la Figura 5.3

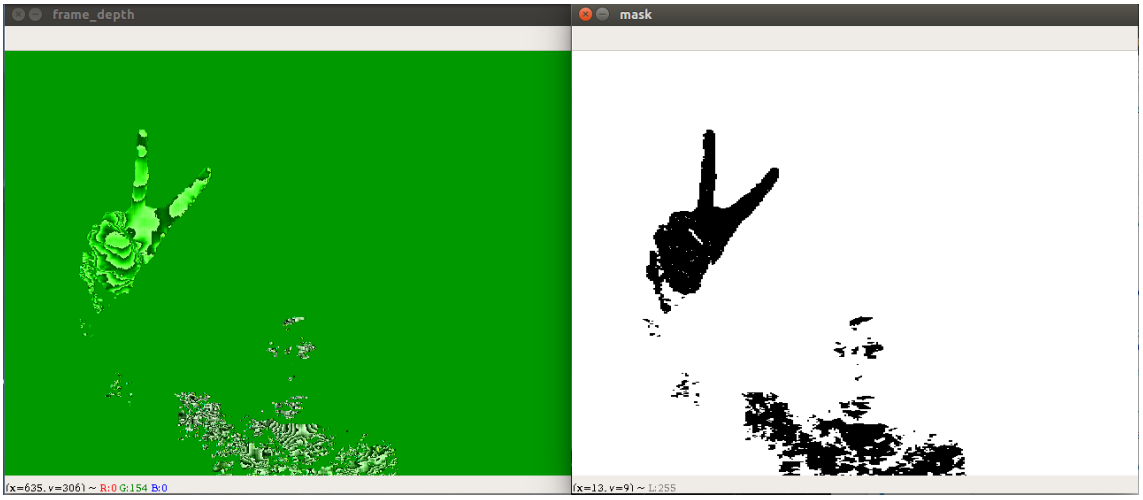


Fig. 5.3 A la izquierda imagen infrarroja sin filtrar. A la derecha misma imagen tras aplicar una máscara

Cómo podemos ver en la imagen de la Figura 5.3, ya tenemos la imagen de la mano en blanco y negro, pero aún aparecen líneas en blanco dentro de la Figura de la mano ya que algunas tenían la misma tonalidad de blanco que el fondo.

Para eliminar estas líneas residuales y suavizar toda la imagen de la mano para que sea uniforme, podemos aplicar un filtro el cual suavice la imagen. El filtro utilizado es el denominado filtro mediana. Este filtro lo que hace es reemplazar cada pixel de la imagen con la mediana de los pixeles vecinos, situados en el cuadrado que rodea a este pixel.

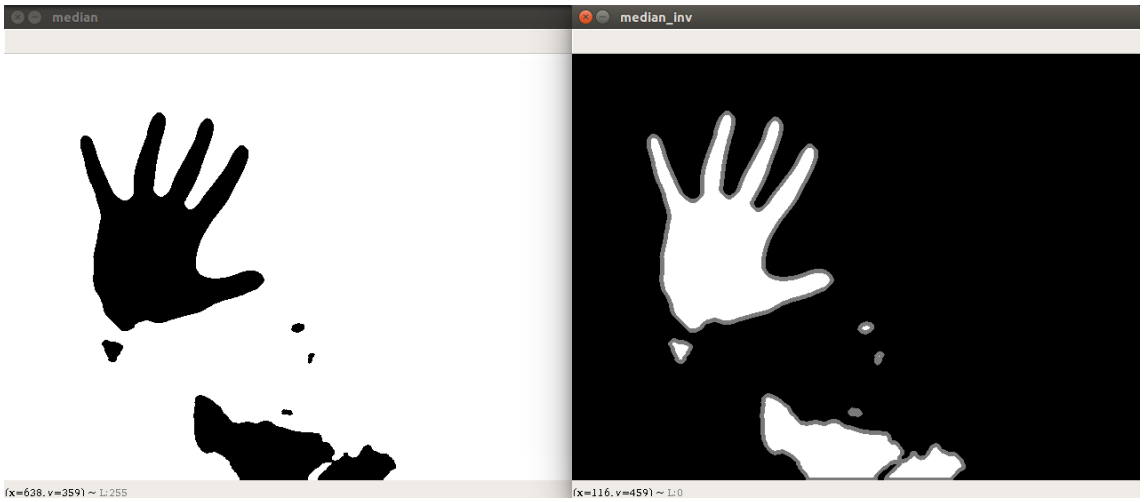
Fig. 5.4 Imagen de una mano antes y después de aplicar el filtro mediana

Tras haber aplicado el filtro ya tenemos una imagen en la que podemos identificar claramente la mano y en la que hemos podido eliminar prácticamente todo lo que no le pertenece.

Una vez hecho esto seguimos sin tener la imagen ideal. Para nuestra aplicación solo queremos la información de la mano, y no todo lo de alrededor, donde nos encontramos con ruido proveniente de nuestro propio cuerpo y con un montón de espacio desocupado. Esto desemboca en dos problemas. El primero es que al utilizar parte de la imagen que no necesitamos, estamos utilizando un mayor coste computacional el cual puede ralentizar de forma muy notoria la ejecución de nuestro programa. En segundo

lugar, nos encontramos con un problema realmente serio a la hora de realizar un entrenamiento ya que dos imágenes que muestren la misma información (mismo número de dedos) pueden convertirse en dos imágenes muy distintas a la vista de nuestro algoritmo si nos las encontramos en zonas distintas de la imagen, o si una está más o menos alejada, aumentando el tamaño proporcional de la mano. Para subsanar estos posibles problemas futuros, aparece la denominada Región de Interés, también comúnmente denominada ROI por sus siglas en inglés (Region Of Interest).

Como su nombre indica, la región de interés consiste en centrarnos solo en la región de la imagen que nos interesa. Existen distintos métodos, pero nosotros nos vamos a aprovechar de las herramientas que pone a nuestra disposición la librería de OpenCV en Python. Lo primero que deberemos hacer es buscar todos los contornos de los objetos que aparezcan en la imagen. En nuestro caso, gracias a la filtración que hemos logrado con la cámara de infrarrojos, así como con los distintos filtros aplicados, prácticamente todo el contorno dibujado pertenecerá a nuestra mano, a excepción de algunos puntos pequeños que hayan aparecido los cuales consideraremos como residuo. Con esto observamos que siempre que mostramos la mano, el contorno más grande de todos pertenecerá a la mano, y el resto de los contornos podrán ser considerados ruido. Con esto podemos hacer una criba y trabajar a partir de ahora solo con el contorno que englobe la mayor área.



.

Fig. 5.5 Imagen antes y despues de dibujar los contornos

En la imagen de la derecha de la Figura 5.5, se puede apreciar cómo se ha añadido una fina línea de tono grisáceo que rodea todos los contornos que aparecen en la imagen. Cabe decir que hemos tenido que invertir los colores de la imagen para que la función otorgada por la librería OpenCV para la captación de contornos funcionase de forma correcta.

Una vez tenemos esto, solo tenemos que crear un rectángulo alrededor del contorno de área máxima, y ya tendremos nuestra región de interés.

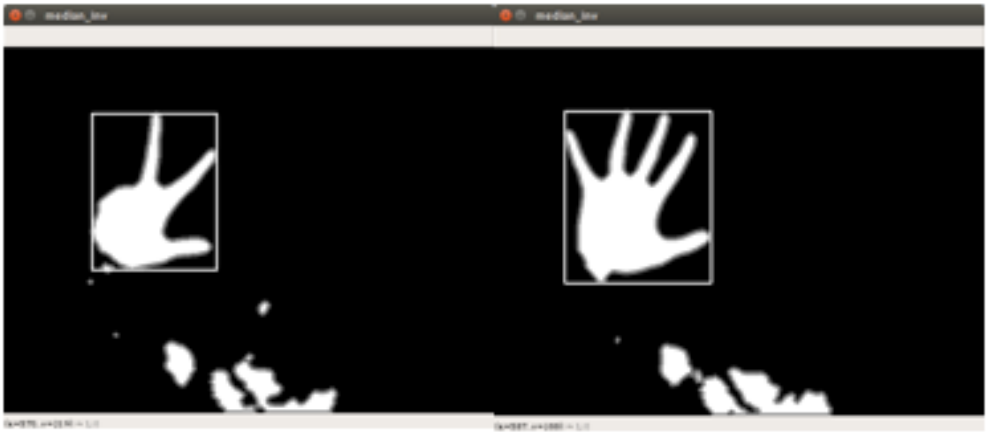


Fig. 5.6 Ejemplos de imágenes mostrando tres y 5 dedos con la región de interés dibujada

En el caso de querer trabajar con dos manos en vez de con una, lo único que tendremos que hacer será declarar también el segundo mayor contorno de la imagen y trabajar con ambos.

Fig. 5.7 Imagen mostrando la región de interés aplicada a dos manos de manera simultánea

Con esto ya habremos definido nuestra región de interés, y habremos separado la imagen de la mano del resto.

Lo último que nos faltaría sería modificar el tamaño de las imágenes, ya que con este método obtendremos imágenes rectangulares de distintos tamaños. Para esto simplemente tendremos que aplicar una función que viene predefinida en la biblioteca OpenCV la cual modifica la imagen al tamaño deseado. Como ya hemos comentado en capítulos anteriores, nosotros optamos por convertir las imágenes en imágenes de 28 por 28 pixeles, ya que tienen resolución suficiente como para poder identificar los dedos de la mano, y no son lo suficientemente grandes como para que afecte de forma notoria al rendimiento de la ejecución. Asimismo, es importante que la imagen sea cuadrada, ya que a la hora de entrenar nuestro clasificador todas las imágenes que le mostremos deberán tener el mismo formato.

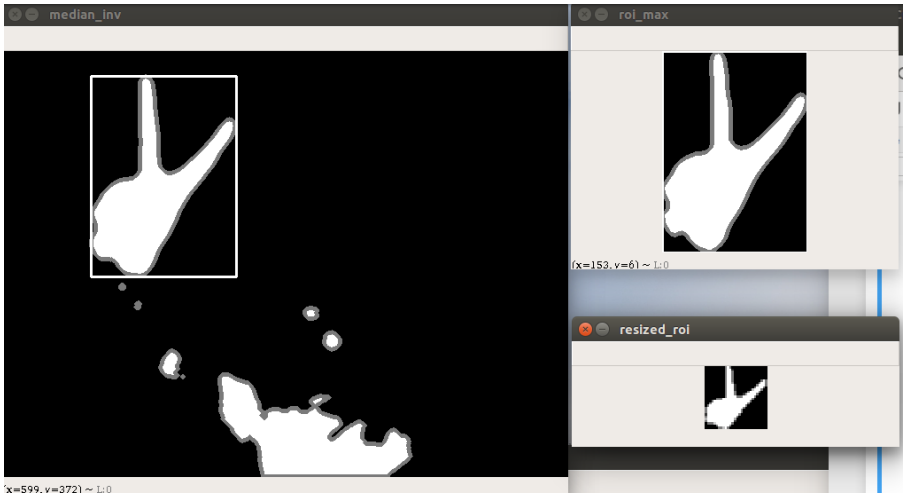


Fig. 5.8 A la izquierda imagen completa mostrando dos dedos. A la derecha arriba imagen de la mano recortada con forma rectangular. A la derecha abajo imagen redimensionada en imagen cuadrada

Con esto ya tenemos unas imágenes lo suficientemente filtradas como para trabajar de forma cómoda, tanto para la adquisición de datos a la hora de realizar nuestro entrenamiento, como para la ejecución de la aplicación a tiempo real.

5.2 Obtención del dataset

Uno de los puntos más importantes a la hora de aplicar un algoritmo de Machine Learning, es la obtención de una buena base de datos con la cual podamos entrenar de manera correcta nuestro clasificador. Primero deberemos pensar si es necesario que nuestros datos estén etiquetados (aprendizaje supervisado), o no es necesario que estén etiquetados (aprendizaje no supervisado). Para ello deberemos pensar que tipo de resultado queremos obtener de nuestra máquina de aprendizaje. Si elegimos aprendizaje supervisado, puede que la obtención sea mucho más tediosa ya que no solo habrá que coger los datos, sino que también habría que etiquetarlos. La parte positiva de este método es que no requiere una base de datos tan grande como podría ser en los métodos no supervisados, ya que, al entrenar al clasificador con los datos etiquetados, este no tendrá que averiguar a qué clase pertenece cada dato. En el caso del aprendizaje no supervisado, la obtención de datos será más sencilla, ya que no tendríamos que preocuparnos del etiquetado, pero como hemos dicho, los resultados pueden ser más confusos en este tipo de aprendizaje.

En nuestro caso la obtención de la base de datos con su etiquetado resultó bastante sencilla. Aprovechando las técnicas de extracción de fondo, detección de los bordes de la mano, obtención de la región de interés, etc., explicadas anteriormente, solo tuvimos que crear un programa en Python que capturase una imagen de la mano al pulsar una tecla del teclado y que automáticamente guardase esa imagen en una carpeta predefinida. Por lo tanto, tomamos una serie de imágenes mostrando distintos números de dedos, y fuimos guardando dichas imágenes en sus correspondientes carpetas. Como hemos dicho, para los métodos supervisados no se necesitan bases de datos gigantescas, y en nuestro caso con menos de 3000 imágenes totales, tendríamos suficientes datos como para poder trabajar con porcentajes de efectividad lo suficientemente altos.

Otro de los puntos importantes a la hora de la realización de nuestra base de datos es la de seleccionar que resolución queremos que tengan las imágenes que vamos a capturar. Esto es importante ya que una resolución demasiado pequeña, puede hacer que los datos que obtengamos no sean representativos y que el clasificador no pueda diferenciar bien las imágenes de una clase con las de otra, pero si la resolución de las imágenes aumenta, el coste computacional puede llegar a ser muy costoso, y hacer que el programa no

funcione con fluidez, o que los tiempos de entrenamiento, predicción, etc. se alarguen un tiempo exagerado. Finalmente decidimos tomar imágenes de 28x28 pixeles.

También es importante a la hora de obtener una buena base de datos es la heterogeneidad. Es importante que nuestra base de datos muestre cierta heterogeneidad. Es decir, es importante que imágenes de la misma clase no sean demasiado parecidas, ya que eso hace que el clasificador no sea capaz de generalizar. El objetivo de la inteligencia artificial es que la maquina sea capaz de generalizar. Si todas las imágenes de un dedo, por ejemplo, son muy parecidas, o tienen la misma orientación, cuando le demos al clasificador una imagen nueva que no sea igual a las que hemos utilizado para el entrenamiento, este no será capaz de generalizar y dará resultados erróneos. Por el contrario, si en la gama de imágenes de un dedo, tenemos un montón de imágenes que sean muy diferentes entre sí, pero que todas representen un dedo, el clasificador generalizara con más agudeza y obtendremos resultados muchísimo mejores. Dicho esto, concluimos con que es mucho más importante tener pocas imágenes, pero que sean diferentes entre sí, a tener muchas imágenes iguales.

Fig. 5.9 A la izquierda seis imágenes mostrando dos dedos heterogéneas entre sí. A la derecha, seis imágenes mostrando dos dedos homogéneas entre sí.

En la imagen de la Figura 5.9 podemos apreciar dos tipos de bases de datos, uno en el cual las imágenes tienen bastantes diferencias y otro en el cual son todas prácticamente iguales. Para la resolución del problema será mucho más efectivo una base de datos lo más heterogénea posible, ya que esto propiciará una mayor generalización.

5.3 Organización de los datos

Una vez tenemos una serie de imágenes de 28*28 pixeles, debemos pensar en cómo de vemos recolocar estas imágenes para que el clasificador pueda trabajar con ellas. Cualquier algoritmo de aprendizaje supervisado, que tenga el fin de hacer una clasificación, va a trabajar siempre con dos matrices. Una de ellas será la matriz en la que están los datos, y la otra es una matriz en la que están sus etiquetas. Las imágenes al final no son más que matrices de números. En nuestro caso, cada píxel es un numero comprendido entre 0 y 255, distribuidos en matrices de 28 filas y 28 columnas. Si nuestra imagen en vez de ser en escala de grises fuese en RGB, por ejemplo, necesitaríamos 3 matrices de 28*28, una matriz para cada color, o lo que es lo mismo matices de tres dimensiones. Por suerte este no es nuestro caso, ya que cuantas más dimensiones tengamos, más se complica el problema.

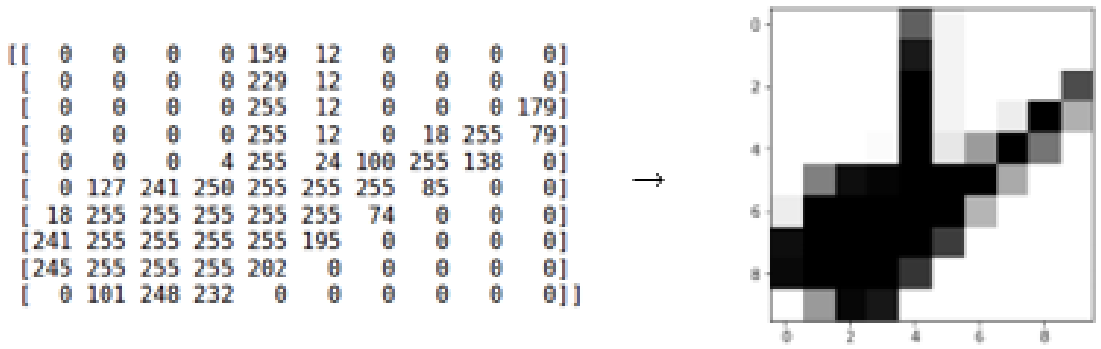


Fig. 5.10 Transformación de matriz numérica en imagen de 8x8 pixeles

Aun así, podemos reducir a un más el número de dimensiones que tenemos y así facilitar el manejo de datos convirtiendo las imágenes en vectores, o lo que es lo mismo en una matriz de una sola dimensión. Es decir, convertimos nuestra matriz de 28*28 en una de 784.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow [1 \;\; 2 \;\; 3 \;\; 4]$$

Fig. 5.11 Conversión de Matriz de dos dimensiones a una dimensión

Una vez obtenidas todas las imágenes vectorizadas, así como todas las referencias de las etiquetas, añadiremos todos sus valores en un Numpy Array. Un Numpy Array no es más que un arreglo especial proporcionado por la biblioteca Numpy de Python el cual es de alta eficiencia y está diseñado principalmente para calculo científico. Finalmente, lo que tendremos serán dos arreglos de este tipo, uno de ellos contendrá todos los vectores con la información de las imágenes, y él otro todas las etiquetas correspondientes a las imágenes.

Con esto ya tendremos las imágenes prácticamente listas para aplicarlas a nuestro clasificador. Solo faltará hacer una diferenciación entre las imágenes que vayamos a utilizar para entrenamiento, y las imágenes que vayamos a utilizar para test.

5.4 Imágenes de entrenamiento e imágenes de prueba

Otra de las partes importantes de nuestro proceso de aprendizaje, es hacer una diferenciación entre las imágenes que vayamos a utilizar para entrenamiento y las imágenes que vayamos a utilizar para realizar las pruebas y pruebas de rendimiento.

Esto puede parecer una tontería, pero es mejor dejarlo claro. Un error gravísimo a la hora de obtener los resultados de nuestro clasificador es hacer la prueba con las mismas imágenes que se han utilizado para el entrenamiento. Si hacemos esto, los más probable es que nuestro clasificador muestre resultados de prácticamente el 100% de efectividad. Este resultado es erróneo, ya que no es lo mismo memorizar, que resolver un problema nuevo. Para explicarlo mejor, podemos hacer una analogía con un examen, por ejemplo. Si yo me estudio un problema, y luego en el examen cae exactamente el mismo problema, y lo resuelvo a la perfección, no quiere decir que sepa hacer el problema, simplemente que lo he memorizado. El objetivo de la inteligencia artificial no es memorizar un problema, por lo que la forma de comprobar si se sabe hacer el problema o no, es hacer el examen con un problema que no haya sido estudiado, y que, con los conocimientos adquiridos durante el estudio o entrenamiento, seamos capaces de resolver ese problema con éxito.

Dicho esto, faltaría por estimar el porcentaje sobre el total de datos que deberemos utilizar para entrenamiento, y el que deberemos utilizar para el test o validación. Ciertamente este no es un número que sea especialmente relevante. Si utilizamos pocos datos de entrenamiento, y muchos datos de test, podemos tener un entrenamiento pobre, y un test que nos pueda dar conclusiones, pues cuanto más amplio sean los datos de test, los resultados que obtengamos serán más precisos. Por el contario si utilizamos muchos datos de entrenamiento y pocos de test, tendremos un clasificador mejor entrenado, pero puede que no tengamos información suficiente para que los resultados obtenidos sean relevantes. Lo que habría que hacer sería intentar buscar una especie de equilibrio que nos dé suficientes datos de entrenamiento como para obtener buenos resultados, y aun así tener datos suficientes para poder sacar conclusiones realistas. Normalmente este porcentaje suele rondar el 80% de datos de entrenamiento y el 20% de datos de test, pero como he dicho es un dato que puede variar y dependerá sobre todo de lo suficientemente grande que sea nuestro dataset.

5.5 Declaración y ajuste del entrenamiento

En el capítulo 3 hemos explicado cómo funcionan por dentro los distintos métodos de Machine Learning que hemos utilizado para nuestro objetivo. En este apartado vamos a centrarnos en cómo se aplican estos métodos a un determinado dataset. Para aplicar los métodos en Python se ha utilizado la librería scikit-learn, la cual es realmente útil ya que, aparte de ser realmente potente, es extremadamente fácil de utilizar. Funciona como una especie de caja negra en la que tu metes unos datos y ella sola te saca todos los resultados sin que tu veas que ocurre dentro. Esto hace que cualquier persona con unas nociones mínimas de matemáticas, algebra y programación, sea capaz de programar un clasificador totalmente valido.

Como hemos dicho scikit-learn funciona como una caja negra, y para hacer todo el ejercicio de entrenamiento solo tendremos que aplicar una pequeña función:

```
clf = MetodoQueQueremosUtilizar(Parametros)
```

Aquí solo tendremos que decirle que método de clasificación que vayamos a utilizar, y enviarle ciertos parámetros, los cuales definimos en el capítulo 3, cuando hablemos de los algoritmos de aprendizaje utilizados, ya que cada algoritmo tiene sus propios

parámetros. El programa automáticamente guardara este método con sus parámetros en la variable *"clf"*. Con esto ya tendremos nuestro clasificador creado, por lo que el siguiente paso será entrenarlo.

5.6 Entrenamiento y predicción

Como ya hemos dicho antes, la librería de scikit-learn es extremadamente intuitiva, y lo único que tendremos que hacer para lleva a cabo el entrenamiento de nuestro clasificador solo tendremos que llamar a un método, y el solo hará todo el trabajo.

```
clf.fit(X_train, y_train)
```

Tras este pequeño comando, nuestro clasificador ya estará listo, y ya podemos introducirle datos para que realice cualquier predicción que queramos.

```
Prediction = clf.predict(X_test)
```

Esto nos devolverá una cadena con las etiquetas que el clasificador ha creído oportunas para los datos que le hemos introducido de test. Con esto podríamos obtener una visión global de cómo está trabajando nuestro clasificador, pero existen métodos mucho mejores para la visualización de estos resultados, y así poder ver dónde están los fallos y donde podemos mejorar, los cuales explicaremos más adelante.

5.7 Elección de los parámetros de entrenamiento

Uno de los puntos importantes a la hora de la elaboración de un clasificador de aprendizaje, será la elección de unos parámetros los cuales maximicen nuestro porcentaje de acierto a la hora de realizar, y a la vez no generen algoritmos demasiado complejos en los cuales podamos tener problemas de sobre entrenamiento.

El método que hemos utilizado es el Grid Search. La idea y las bases de este método son bastante simples, pero a la vez es realmente efectivo. El procedimiento consiste en crear una variable con distintos valores para los parámetros, y hacer tanto el entrenamiento como el test, con todas y cada una de las combinaciones posible.

Después de esto compararemos todos los resultados y nos quedaremos con la combinación óptima El problema que tiene este método es su largo tiempo de procesado, ya que, si tenemos muchos parámetros o muchos valores diferentes, la obtención de resultados puede demorar hasta varias horas.

Otra alternativa a este método en el que combinamos todas las opciones posibles es el de realizar combinaciones aleatorias de los parámetros y sus valores. Esta opción sería mucho más rápida, pero los valores que obtendríamos no serían tan precisos como en el primer método. En nuestro caso hemos decidido utilizar el primer método, ya que, aunque pueda ser mucho más lento, este cálculo solo se ha de realizar una sola vez y una vez tengamos unos parámetros con un grado alto de efectividad, trabajaremos siempre con estos de aquí en adelante.

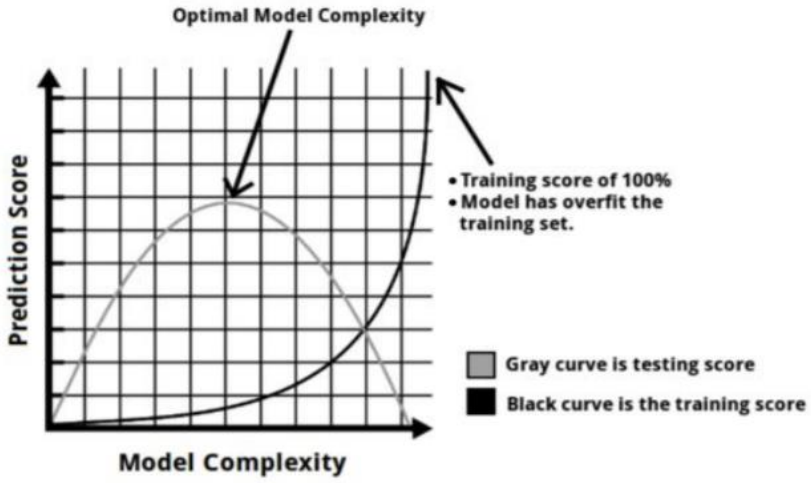


Fig. 5.12 Sobre entrenamiento

En la gráfica de la Figura 5.12 podemos ver claramente el problema del sobre entrenamiento. Si nos fijamos en la línea negra, la cual marca la puntuación obtenida sobre los mismos datos con los que se ha entrenado el clasificador, no deja de aumentar a medida que nuestro clasificador es más complejo. Esto es así porque nuestro clasificador ha dejado de pensar, y está empezando a memorizar, por lo que solo será capaz de identificar los datos que sean exactamente iguales a los del entrenamiento. En el caso de la curva gris, el punto óptimo se encuentra en un punto de equilibrio entre la generalización y la complejidad del algoritmo.

5.8 Obtención y representación de resultados

Esta es la parte final a la hora de hacer un clasificador, y es muy importante ya que es la que nos dice si nuestro clasificador está trabajando adecuadamente, o si en caso contrario, deberíamos volver a dar un repaso al algoritmo completo para poder ver donde estamos fallando y donde debemos mejorar.

Existen diversos datos para cuantificar el error cometido con nuestro clasificador, y aquí voy a mostrar los que me han resultado más útiles a la hora de realizar mi algoritmo.

5.8.1 Valores porcentuales

Lo mejor de este método es que es muy rápido de obtener y es un dato que todo el mundo entiende a primera vista. Nos puede venir muy bien para ver una primera aproximación de cómo está trabajando nuestro clasificador. Simplemente tendremos que aplicar la siguiente ecuación:

$$Precision = \frac{Numero\ de\ Predicciones\ Correctas}{Numero\ Total\ de\ Datos\ Introducidos} \qquad \textit{Ecuación 5.1}$$

Esta ecuación predice los datos que hemos introducido en X_test, y comparará las etiquetas que haya predicho, con las que tenemos guardadas en y_test. Esto nos devuelve un número decimal entre 0 y 1 que equivale al porcentaje de acierto, siendo uno el 100% de acierto y 0 el 0%.

Es importante decir que este método no siempre es el más correcto. Solo será correcto si nuestros datos están equilibrados, es decir, si tenemos aproximadamente el mismo número de imágenes de 1, 2, 3, 4 y 5 dedos. Veamos la explicación con un ejemplo. Imaginamos que entrenamos a nuestro clasificador solo con imágenes de uno y dos dedos para hacer más sencilla la explicación. Introducimos a nuestros datos para el test un 60% del total de imágenes de 1 dedo, y el 40% de imágenes de dos dedos. Si después de hacer el cálculo de la precisión obtenemos un valor del 95% de acierto, podemos

decir que nuestro clasificador está trabajando bastante bien. Por el contrario, supongamos que ahora nuestra base de datos de test está compuesta por 100 imágenes, de las cuales 95 son imágenes de un dedo y las otras 5 son de 2 dedos. Imaginemos ahora que nuestro clasificador no está trabajando correctamente, e identifica todas las imágenes como imágenes de un dedo. Nuestro clasificador ahora seguiría teniendo un porcentaje de acierto del 95%, pero en este caso no podemos decir que esté trabajando de forma correcta.

5.8.2 Cross-Validation

A este método, también denominado validación cruzada, lo podríamos considerar como una evolución del método que utilizábamos para calcular el porcentaje de acierto. Con los métodos anteriores, estábamos realizando todo el tiempo el entrenamiento y la validación con los mismos datos, lo que puede suponer un error metodológico.

En este método lo que se trata es de jugar con el dataset completo, en concreto consiste en hacer diferentes divisiones entre datos de validación y datos de entrenamiento, para hacer el examen con todas las opciones y luego hacer una media de los resultados obtenidos. Con esto obtenemos un resultado que es mucho más real al obtenido realizando solo una vez el experimento

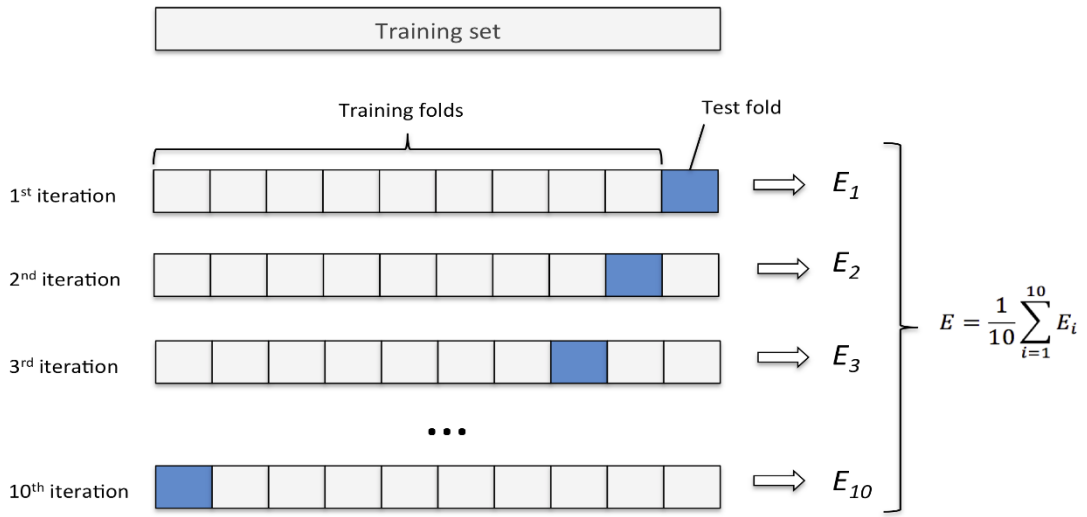


Fig. 5.13 Esquema Cross-Validation [41]

En el ejemplo de la Figura 5.13, separamos nuestro dataset de forma que un 90% se utilizará para el entrenamiento y un 10% para test. Con esto podremos realizar 10 veces el entrenamiento, y 10 veces el cálculo del porcentaje de acierto.

5.8.3 Matriz de confusión

Esta es la herramienta más utilizada para para evaluar cualquier algoritmo de clasificación, ya que, a pesar de su sencillez, nos da una gran cantidad de datos útiles. La Figura 5.14 muestra el ejemplo más simple de una matriz de confusión, pero que a su vez es el que mejor la define.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	TN	FP
	Positive	FN	TP

Fig. 5.14 Esquema básico de una matriz de confusión

En ella se comparan las etiquetas que se han predicho, con las etiquetas reales que tienen los datos, siendo:

- TN (True Negatives): Verdaderos Negativos. Valores que se han predicho como negativos, y en efecto eran negativos.
- FP (False Positives): Falsos Positivos. Valores que se han predicho como negativos y por el contrario eran positivos.
- FN (False Negatives): Falsos Negativos. Valores que se han predicho como negativos y por el contrario eran positivos.
- TP (True Positives): Verdaderos Positivos. Valores que se han predicho como positivos y en efecto eran positivos.

Viendo esto, podemos interpretar que cuanto mayor sean los números que se encuentran en la diagonal TN-TP comparados con el resto de números, mejor estará funcionando nuestro clasificador.

En este caso hemos visto una matriz de confusión para valores binarios, en los que solo tenemos positivos y negativos, pero esto se puede extrapolar a modelos de clasificación multi-clase como es nuestro caso. En el ejemplo de la Figura 5.15, podemos ver una matriz de confusión para 5 clases. Podemos ver en la escala de colores que cuanto más claro sea el color de la diagonal mejor está funcionando nuestro clasificador.

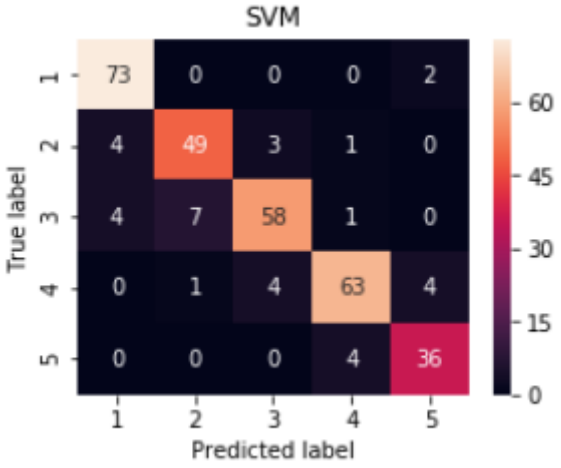


Fig. 5.15 Ejemplo de matriz de confusión

El problema que pueden tener estas matrices es que, si los datos no están equilibrados, podemos crearnos una pequeña confusión. Cuando decimos que no está equilibrada en este caso, nos referimos a que tenemos más cantidad de imágenes de un tipo que de otro en nuestro dataset. En el ejemplo de la Figura, podemos ver por ejemplo que, para la etiqueta del 5, tenemos 36 verdaderos positivos y para la del uno, tenemos 73. Esto no quiere decir que nuestro clasificador este realizando un mejor trabajo para el uno que para el 5, solo quiere decir que tiene más datos de test para el uno, porque el número total será más alto.

Gracias a la matriz de confusión, también podemos ver de forma bastante clara, en que clases cometemos más fallos. En la imagen de la Figura 5.16, por ejemplo, podemos ver que estamos confundiendo demasiadas veces el dos y el cuatro. Si nos fijamos, tenemos

60 casos en los cuales deberíamos haber predicho un cuatro y hemos predicho un dos, y lo mismo ocurre en el caso contrario, en el cual tenemos 53 errores. Esto comparado con el total de muestras, es una señal de que algo está fallando con estos dos números y habrá que prestar mayor atención y hacer algún ajuste a nuestro clasificador o a nuestra base de datos.

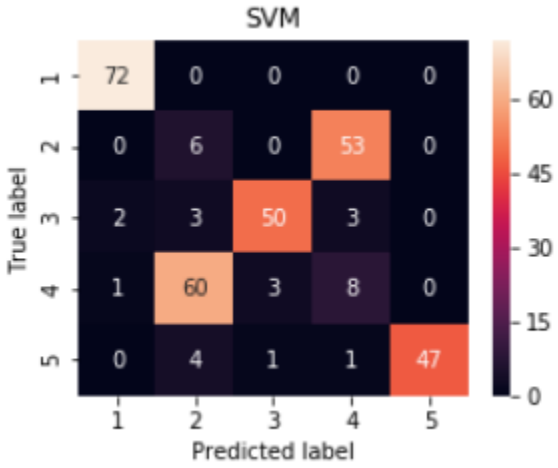


Fig. 5.16 Ejemplo de matriz no equilibrada

En la práctica, hemos utilizado los tres tipos de evaluación. Para la obtención de los parámetros utilizamos una combinación de los valores porcentuales y la validación cruzada, y para la evaluación más exhaustiva utilizamos las matrices de confusión.

5.9 Predicción en tiempo real

Finalmente, tras haber realizado todos los puntos anteriores, y haber comprobado que podemos tener una máquina de clasificación, que puede ser válida, viene la parte final, la simulación en tiempo real. Esta parte es realmente sencilla, ya que lo único que tendremos que hacer será realizar un bucle en el cual introducimos continuamente las imágenes que estamos captando por la cámara a nuestro predictor, y mostrar los resultados por pantalla. El único problema que podríamos tener es que si, o bien nuestro algoritmo es muy complejo, o las imágenes que estamos tomando tienen demasiada resolución, el procesador de nuestro computador podría saturarse y no ejecutar nuestro programa de forma óptima. En nuestro caso, utilizando un ordenador de gama media-alta, no tenemos ningún tipo de problema de procesado, y el programa fluye de forma correcta.

Arriba a la izquierda se muestran las predicciones que está realizando cada uno de los clasificadores diseñados como en lo ejemplos de la figura 5.17.

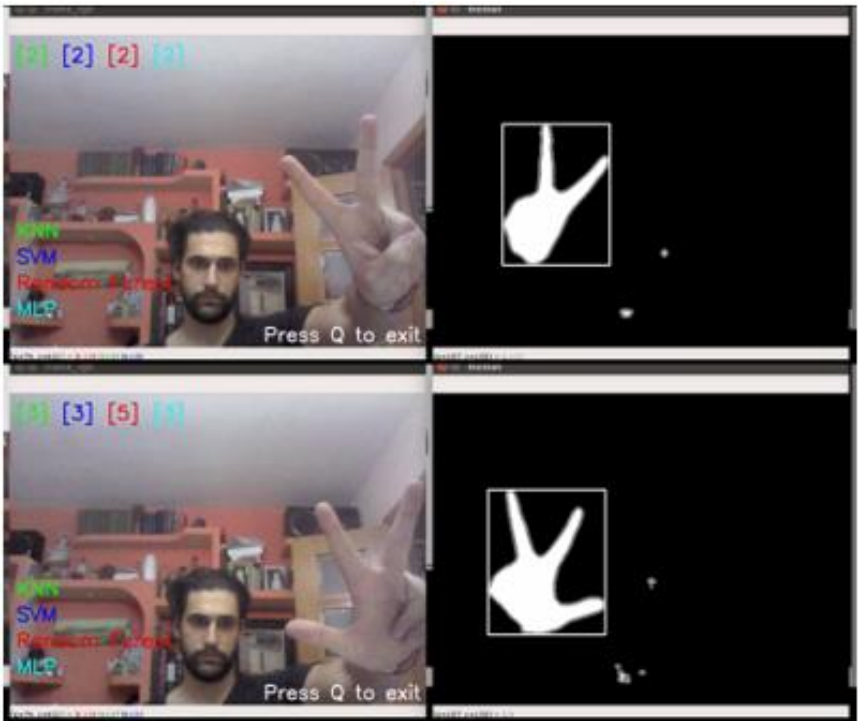


Fig. 5.17 Detectando dedos en tiempo real

5.10 Juego de pares y nones

Como punto final del proyecto, se decidió aplicar los resultados obtenidos en un simple juego para demostrar una de las posibles aplicaciones que podría tener en el mundo de la interacción entre el hombre y la máquina.

Este juego es el famoso juego de pares y nones. Funciona de forma muy sencilla. Se ejecuta el archivo de juego, y se mostrará la imagen captada por la cámara RGB en la que nos veremos a nosotros mismos. Si pulsamos la tecla S del teclado el juego dará comienzo, mientras que, si pulsamos la tecla Q, saldremos de la ejecución. Una vez el juego inicia, deberemos enseñar la mano a la cámara y se tomará una captura. El programa generará a su vez un número aleatorio entre el 0 y el 5. Si la suma de nuestra mano y la del ordenador suman pares, habremos ganado. Por el contrario, si la suma es impar, habremos perdido. Para jugar otra partida deberemos pulsar la tecla R.

Fig. 5.18 Pantalla principal del juego

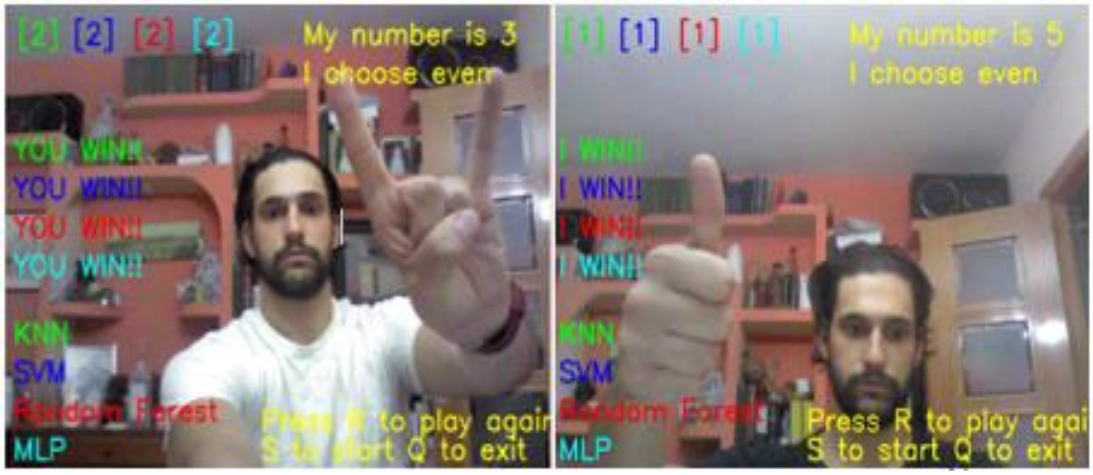


Fig. 5.19 Ejemplos de partidas de pares y nones

6 RESULTADOS

En este capítulo nos vamos a centrar en el análisis de los resultados obtenidos aplicando los distintos métodos estudiados sobre nuestra base de datos. Primero realizaremos un análisis utilizando las imágenes de la base de datos, y luego realizaremos otro análisis comparando todos los métodos en la ejecución en tiempo real.

El procedimiento seguido será el que ya hemos explicado en capítulos anteriores. Primero obtendremos los parámetros mediante el método ya explicado de Grid Search. Utilizando este método obtendremos unos porcentajes de acierto los cuales aplican el concepto de validación cruzada. Una vez tengamos estos parámetros, los aplicaremos para realizar de nuevo el entrenamiento y obtener las matrices de confusión.

6.1 Resultados de bosques aleatorios

Empezaremos con la obtención de los resultados del método de bosques aleatorios.

En el caso de los bosques aleatorios, dos de los parámetros, hacen que nuestro clasificador obtenga mejores resultados a medida que aumentan. Estos parámetros son el de “n_estimators” y el de “max_depth”. Esto es así porque estos parámetros hacen nuestro clasificador mucho más complejo a medida que son mayores. En este caso especial, lo que haremos será generar una gráfica con un amplio rango de estos valores y veremos cómo se comportan.

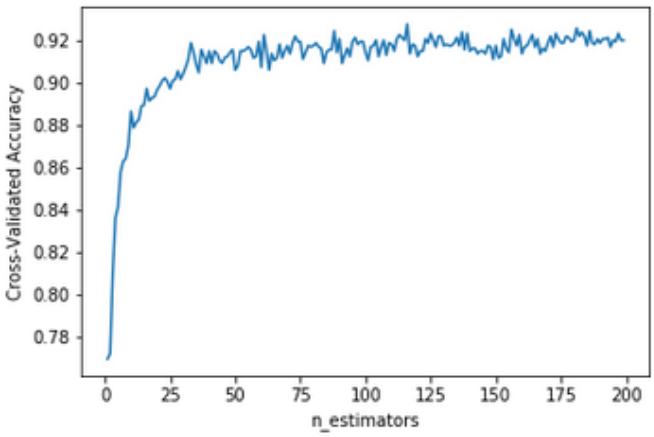


Fig. 6.1 Puntuación dependiendo de n_estimators entre 0 y 200

Para el parámetro hemos seleccionado un rango desde 1 hasta 200, y se han calculado el porcentaje de acierto utilizando una validación cruzada de grado 5. En la gráfica podemos observar que se comporta de manera similar a una función logarítmica, por lo que para números grandes se termina estabilizando. Lo que haremos en este caso será seleccionar el valor más bajo en el cual ya se puede considerar estable. Hay que considerar que cuanto mayor sea este valor, más complejo será nuestro árbol, lo que conlleva a un mayor tiempo de procesado, así como correr el riesgo de sufrir sobre entrenamiento. Viendo esto, seleccionaremos algún valor que ronde los 50. Cabe decir que, en el peor de los casos, en el cual "n_estimators" es igual a uno, el porcentaje de efectividad es de prácticamente un 80%, lo cual está realmente bien.

Veamos ahora con la variable "max_depth".

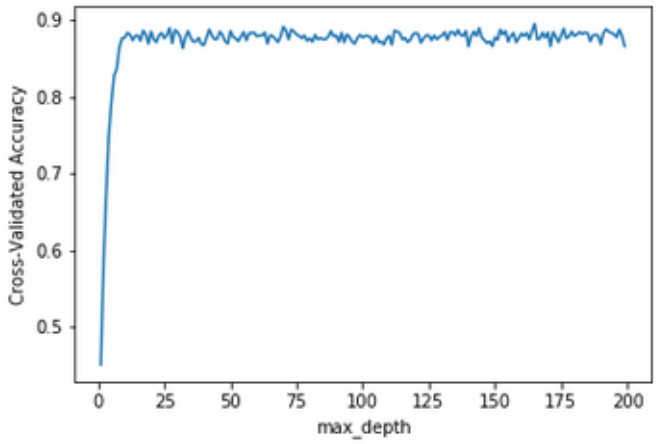


Fig. 6.2 Puntuación dependiendo de max_depth entre 0 y 200

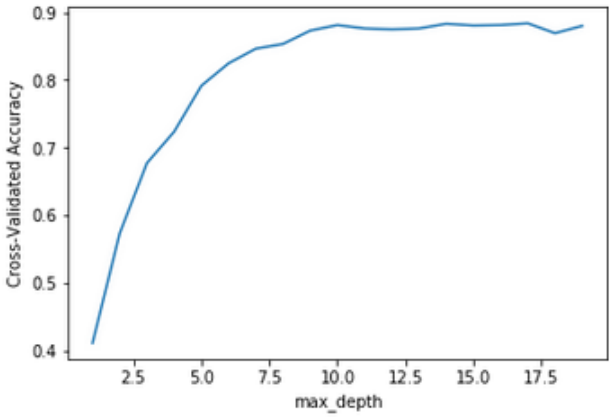


Fig. 6.3 Puntuación dependiendo de max_depth entre 0 y 18

En este caso hemos hecho una primera estimación con valores de 0 a 200 al igual que en el caso de “n_estimators”, y al ver el cambio brusco que aparece lo hemos reducido a un rango de 1 a 20 para poder ver la curva más de cerca. Al igual que en el primer parámetro, aquí también nos encontramos con una gráfica similar a la función logarítmica, por lo que la analizaremos de forma similar. En ella podemos ver que aumenta de forma bastante pronunciada entre el uno y el diez pasando de un 40% hasta prácticamente un 90%. Para este caso, por lo tanto, nos quedaremos con los valores que ronden el 10.

Después de haber visto estos parámetros, vamos a aplicar Grid Search para ver los resultados conjuntos, tanto de estos parámetros, como del resto. Las combinaciones de parámetros que se han utilizado son todas las combinaciones posibles entre los siguientes valores:

- n_estimators: 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75
- criterion: Gini, Entropy
- max_depth: 5, 6, 7, 8, 9, 10, 11, 12

La mejor combinación obtenida de estos valores es:

- n_estimators: 75
- criterion: Entropy
- max_depth: 12

Con un porcentaje de 93.2 y con un grado de validación cruzada de 5. Como era de esperar, los resultados más favorables han sido los que constan con las variables ‘max_depth’ y ‘n_estimators’ más altas.

Pasemos ahora a ver la matriz de confusión aplicando los parámetros obtenidos. Para la matriz de confusión se han utilizado el 80% de las 1574 imágenes de nuestra base de datos para el entrenamiento, y el otro 20% como test.

```
Score Random Forest:  93.94904458598727 %

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=12, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=75, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

Fig. 6.4 Todos los parámetros seleccionados en Bosques Aleatorios

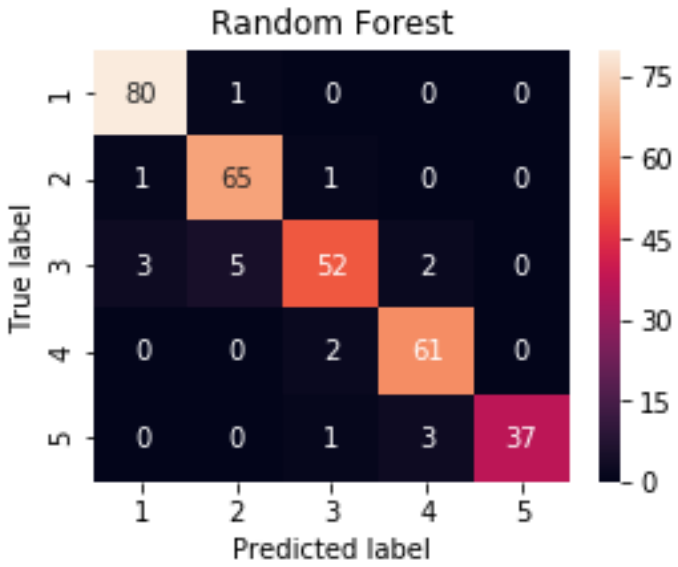


Fig. 6.5 Matriz de confusión de Bosques Aleatorios

En la Figura 6.4 se muestran todos los parámetros con los que se ha entrenado el clasificador.

En la matriz de confusión podemos observar que los resultados son realmente buenos, ya que, de 313 comprobaciones, solo hemos obtenido 19 errores, lo que suma casi un 94 por ciento de acierto.

Con estos resultados podemos decir que este clasificador podría ser totalmente válido para una aplicación real, aunque lo comprobaremos más adelante en la simulación en tiempo real.

6.2 Resultados de máquinas de vectores de soporte

Pasamos ahora a analizar el segundo método estudiado, las máquinas de vector de soporte. En este caso los valores de los parámetros 'C', 'gamma' y 'degree' varían mucho dependiendo del kernel con el que estemos trabajando. Por esta razón, primero analizaremos estos datos para cada kernel de forma aislada y luego haremos un estudio general para ver con que parámetros nos quedaremos finalmente.

6.2.1 Linear

En el caso del kernel lineal, el único parámetro que tenemos para modificar será el parámetro C. Como solo estamos analizando un valor, la mejor forma de visualizar el resultado es de forma gráfica.

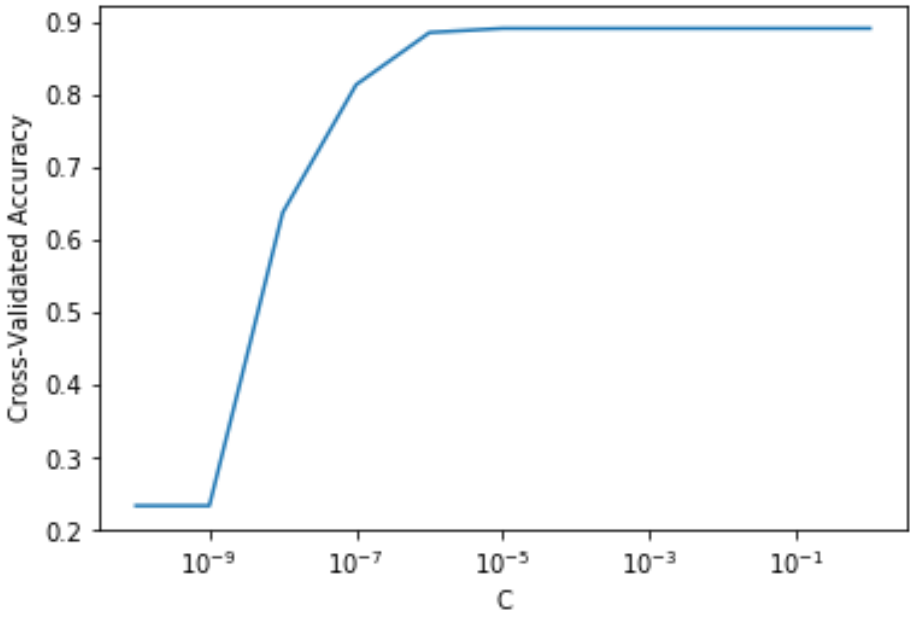


Fig. 6.6 Valores de C en un Kernel lineal

En la gráfica podemos ver como se estabiliza cerca del 90 por ciento para valores mayores de 10^{-6}.

6.2.2 Poly

En este caso tendremos 3 parámetros adicionales los cuales son el valor de C, la constante gamma y el grado de la función. En este caso, como tenemos más de un parámetro, lo más cómodo será aplicar el método de Grid Search.

Tras realizar la búsqueda de los parámetros, los mejores resultados se han obtenido con los siguientes valores:

- c: 1^{-12}
- gamma: 0.1
- degree: 3

Con un porcentaje de acierto de un 95.044%. Si nos fijamos en la gráfica de la Figura 6.6, el mayor porcentaje de acierto es aproximadamente de un noventa por ciento, por lo que utilizaremos la transformación polinómica en la aplicación final.

Estos son los valores que utilizaremos para la aplicación en tiempo real. Veamos ahora como queda la matriz de confusión utilizando estos parámetros.

```
Score SVM:  95.22929993630573 %

SVC(C=1e-12, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='poly',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Fig. 6.7 Parámetros seleccionados en máquinas de vectores de soporte

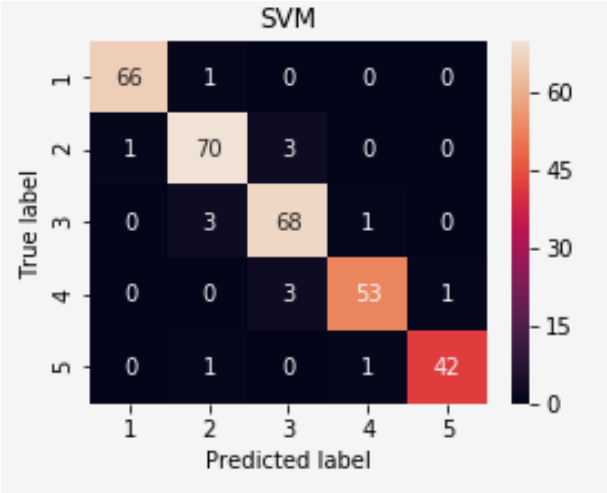


Fig. 6.8 Matriz de confusión de máquinas de vectores de soporte

Viendo los resultados claramente podemos decir que este clasificador es totalmente válido, ya que los errores que tenemos son mínimos.

6.3 Resultados de K-Vecinos más cercanos

Veamos ahora los resultados de el algoritmo de K-vecinos más cercanos. En este caso, la elección de los parámetros no parece compleja de primeras, ya que solo tenemos un parámetro que ajustar. Para ver cómo se comporta lo mejor es ver una gráfica que relacione la precisión con los distintos valores para la K.

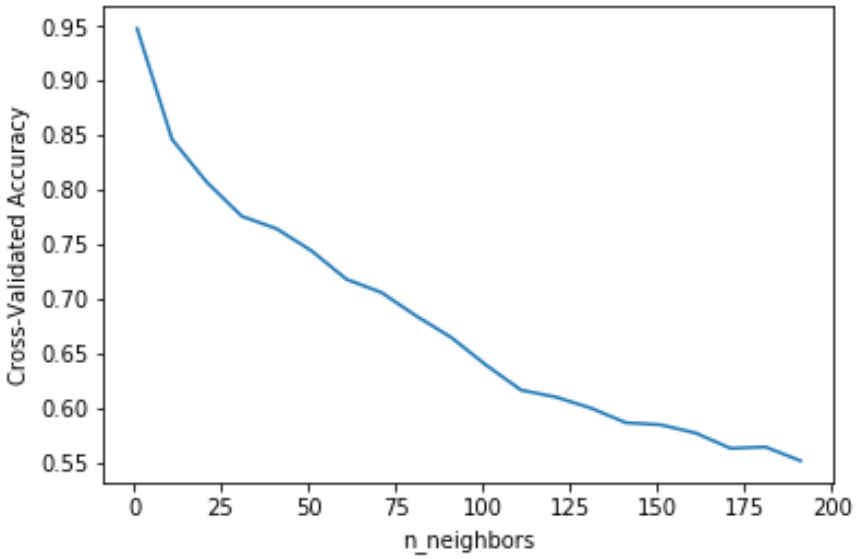


Fig. 6.9 Puntuación dependiendo del número de vecinos

Aquí vemos que la puntuación disminuye muchísimo a medida que aumentamos el grado de vecinos con el que nos estamos comparando. Esto es porque cuanto más pequeña la constante que estamos estudiando, más complejo se convierte nuestro algoritmo, y como las imágenes que tenemos de entrenamiento y de test son bastante similares, es probable que suframos problemas de sobre entrenamiento si cogemos una variable demasiado baja. En este problema en concreto, la mejor forma de probar los distintos parámetros será durante la simulación en tiempo real, ya que al solo tener que modificar un solo parámetro, no nos llevará mucho tiempo su análisis.

De todos modos, veamos algunas matrices de confusión con los distintos valores, para ver su comportamiento.

```
Score KNN:  96.4968152866242 %

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Fig. 6.10 Parámetros completos K-vecino más cercano con n_neighbors=1

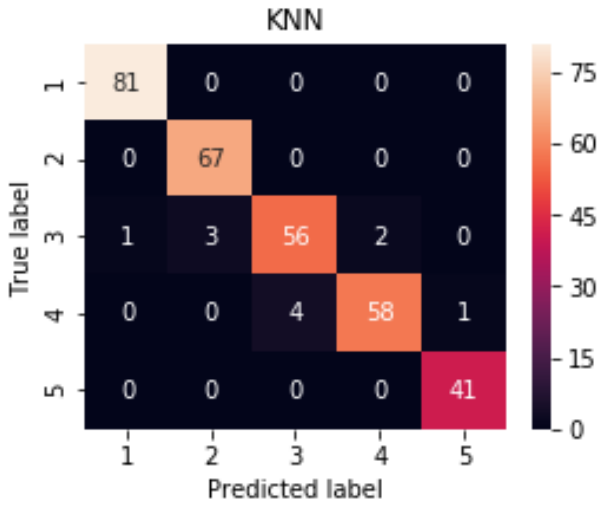


Fig. 6.11 Matriz de confusión K-vecino más cercano con n_neighbors=1

```
Score KNN:  91.40127388535032 %

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

Fig. 6.12 Parámetros completos K-vecino más cercano con n_neighbors=5

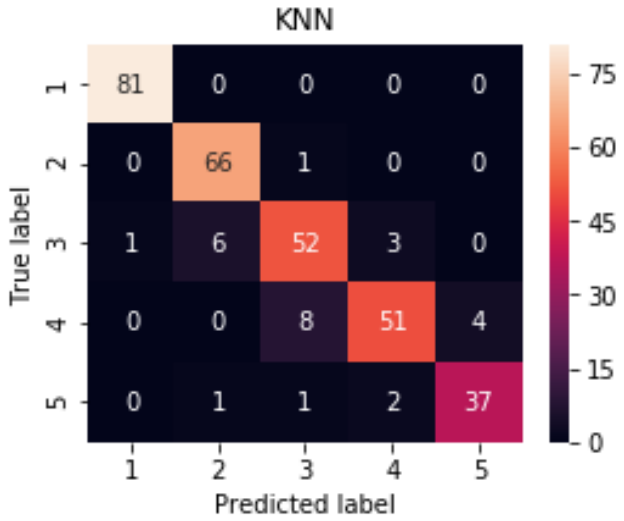


Fig. 6.13 Matriz de confusión K-vecino más cercano con n_neighbors=5

```
Score KNN:  87.26114649681529 %

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=10, p=2,
           weights='uniform')
```

Fig. 6.14 Parámetros completos K-vecino más cercano con n_neighbors=10

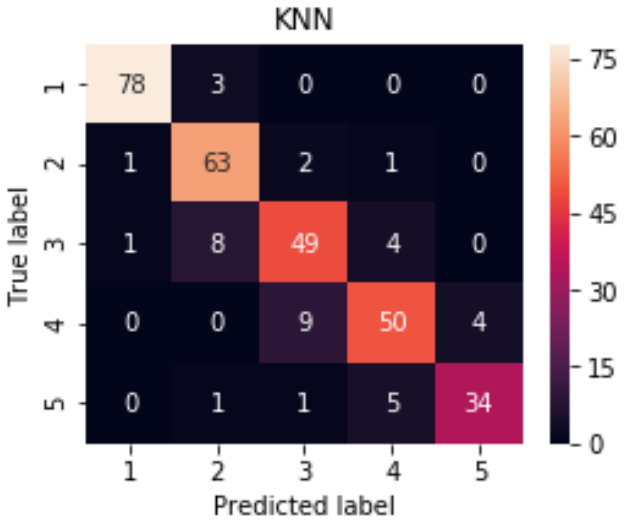


Fig. 6.15 Matriz de confusión K-vecino más cercano con n_neighbors=10

```
Score KNN:  82.16560509554141 %

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=20, p=2,
           weights='uniform')
```

Fig. 6.16 Parámetros completos K-vecino más cercano con n_neighbors=20

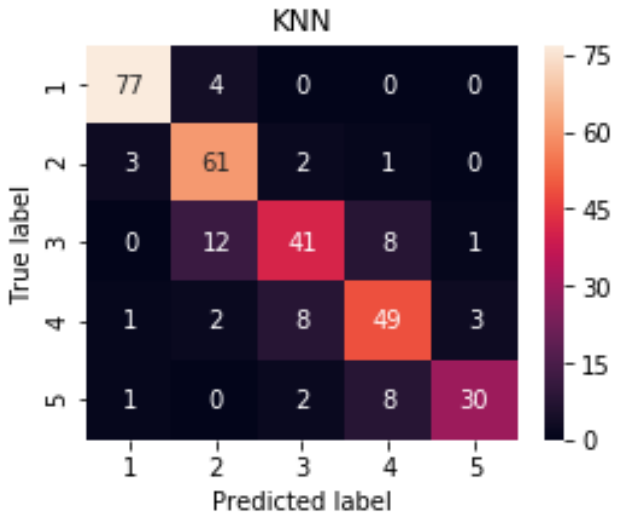


Fig. 6.17 Matriz de confusión K-vecino más cercano con n_neighbors=20

En las matrices podemos comprobar lo que veníamos diciendo anteriormente. A medida que aumenta el área de vecinos con la que nos estamos comparando, peores resultaos aparecen. En la ejecución en tiempo real, deberemos comprobar si con la constate igual a uno trabaja bien, o si, por el contrario, nuestro algoritmo está sobredimensionado.

6.4 Resultados de perceptrón multicapa

Por último, antes de analizar los resultados en la ejecución en tiempo real, vamos a analizar los resultados obtenidos para el algoritmo del perceptrón multicapa.

En este caso pasaremos directamente todos los parámetros y sus combinaciones posibles por el algoritmo de Grid-Search. Con esto podremos estimar los parámetros no numéricos con los que vamos a trabajar. Una vez hayamos decidido estos dos

parámetros, probaremos a introducir en una gráfica el parámetro numérico que nos resta para ver cómo se comporta.

Los parámetros comprobados, y sus correspondientes combinaciones son:

- hidden_layer_sizes: 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400 y 1500
- activation: logistic, tanh, relu

Para analizar los resultados obtenidos veamos una gráfica que compare las tres funciones de activación.

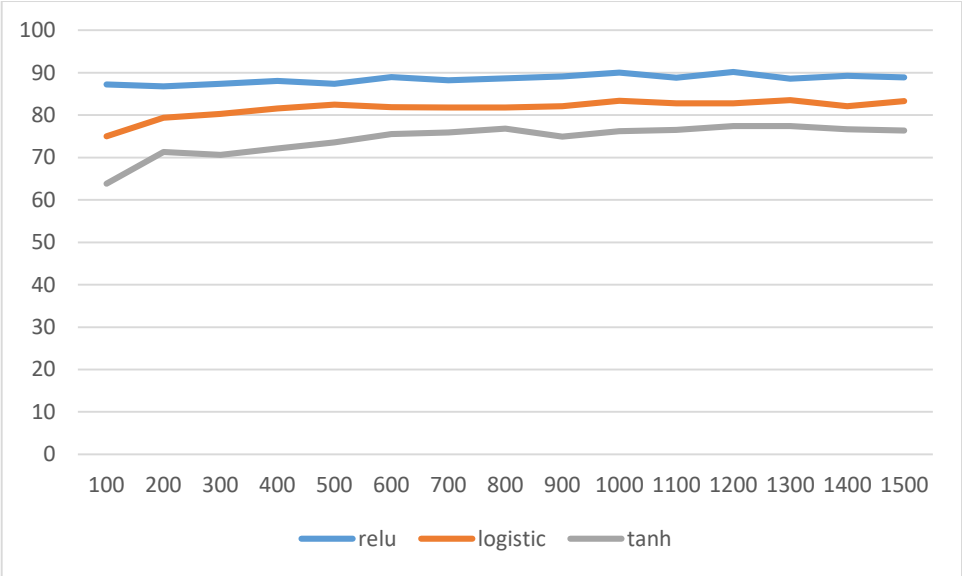


Fig. 6.18 Resultados perceptrón multicapa

En la tabla podemos ver claramente que tenemos los porcentajes más altos de acierto cuando utilizamos 'relu' como método de activación. A su vez, tras realizar el análisis con el método Grid-Search, y aplicando un grado de validación cruzada de 5, obtenemos que el mejor resultado es el siguiente:

- hidden_layer_sizes = 1200 y activation = relu

Con un porcentaje de acierto de un 90.2%

Por lo tanto, estos serán los valores de los parámetros que seleccionaremos para la aplicación. Pasemos ahora a ver cómo quedaría nuestra matriz de confusión con los parámetros seleccionados.

```
Score Multi Layer Perceptron:  88.21656050955414 %

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=1200, learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

Fig. 6.19 Parámetros seleccionados en perceptrón multicapa

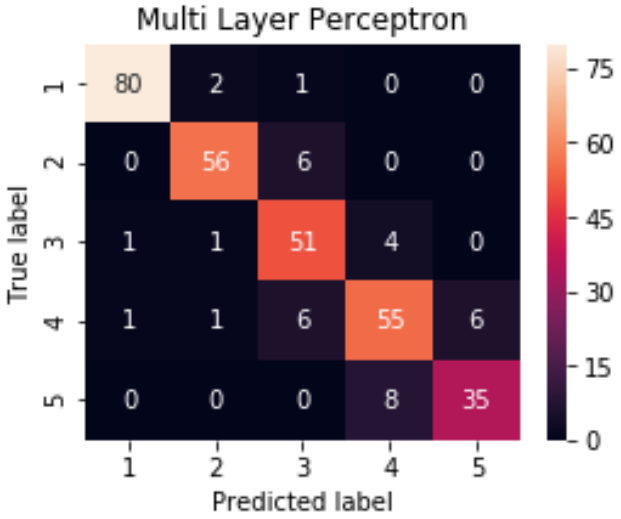


Fig. 6.20 Matriz de confusión de perceptrón multicapa

Al igual que en los casos anteriores, si nos fiamos de los resultados obtenidos con las imágenes de test, podremos decir que nuestro clasificador basado en el algoritmo del perceptrón multicapa, trabaja de forma fiable. Deberemos analizarlos en la simulación en tiempo real para confirmar que trabaja bien.

6.5 Resumen de resultados sobre imágenes de la base de datos

	Porcentaje de acierto Utilizando datos de entrenamiento			
Método	K-NN (K=5)	SVM	Bosques Aleatorios	Perceptrón
1 Dedo	100,00%	98,51%	98,77%	96,39%
2 Dedos	98,51%	94,59%	97,01%	90,32%
3 Dedos	83,87%	94,44%	83,87%	89,47%
4 Dedos	80,95%	92,98%	96,83%	79,71%
5 Dedos	90,24%	95,45%	90,24%	81,40%
Promedio	90,71%	95,20%	93,34%	87,46%

Tabla 2 Resultados con datos de entrenamiento

En la Tabla 2 se recogen todos los resultados de nuestros clasificadores, tras realizar el test sobre los datos de entrenamiento que tenemos de nuestro dataset y utilizando los parámetros que hemos seleccionado en apartados anteriores. En verde están resaltados los resultados más favorables, mientras que en rojo vemos representados los más desfavorables.

En la tabla podemos ver que de todos nuestros clasificadores podrían ser perfectamente válidos, ya que muestran un promedio de acierto de un 85%. Sin lugar a duda, el clasificador con el que obtenemos mejores resultados es con el de las máquinas de vectores de soporte, ya que llegamos a superar un 95% de acierto, y en ninguno de los casos baja del 90% por ciento.

Puede resultar sorprendente que el perdedor en este caso sea el perceptrón multicapa, ya que es el método que se considera más avanzado dentro de los que se aquí se han estudiado. La razón de estos resultados probablemente sea porque este método, y en general todos los métodos basados en redes neuronales, requieren de bases de datos realmente extensas, mientras que nosotros hemos intentado realizar este proyecto con una base de datos pequeña.

En el caso de los bosques aleatorios, los resultados son realmente buenos, exceptuando el caso de las imágenes de tres dedos, en el cual ha obtenido un resultado muy bajo en comparación con el resto.

Por último, el valora de k que hemos considerado para realizar la tabla, es 5, ya que, a juzgar por los resultados, parece que generaliza bastante bien, y no es tan restrictivo como si este valor fuese un uno.

6.6 Resultados de la ejecución en tiempo real

El análisis en tiempo real es un poco más complejo de analizar, ya que no tenemos ninguna forma de ver unos resultados de forma numérica. Nosotros mismos somos los únicos que sabemos a ciencia cierta la cantidad de dedos que estamos mostrando a la cámara en cada momento, por lo que el algoritmo no tiene ninguna forma de comprobar si las suposiciones que está realizando son correctas o no.

Lo que haremos será dar nuestra opinión sobre que clasificador trabaja mejor de la forma más objetiva posible.

Como habíamos comentado en el apartado de resultados del método de k-vecino más cercano, no estábamos del todo seguros de que valor de k íbamos a utilizar para la realización de la simulación en tiempo real. Tras realizar diferentes pruebas utilizando distintos valores de este parámetro, el valor con el que mejor nos ha parecido que trabaja es para k igual a 5.

En la Tabla 3 se recogen los resultados obtenidos en la simulación en tiempo real. En ella hemos hecho una valoración de 1 a 5 significando 1, que predice muy mal, y 5 que predice muy bien. En verde se recogen los mejores resultados y en rojo los peores resultados.

	Valoración sobre la simulación en tiempo real			
Método	K-NN (K=5)	SVM	Bosq. Aleat.	Percep.
1 Dedo	5	5	4	4
2 Dedos	4	4	3	2
3 Dedos	3	4	2	3
4 Dedos	3	3	4	3
5 Dedos	5	5	4	4
Promedio	4,0	4,2	3,4	3

Tabla 3 Resultados de la simulación en tiempo real

En este caso los resultados obtenidos son muy parecidos a los obtenidos en la Tabla 2, con los datos de la base de datos. En este caso damos como ganador a las máquinas de vectores de soporte, las cuales han trabajado de forma sorprendentemente buena, siendo capaces de generalizar mejor que el resto de los clasificadores.

El resultado del perceptrón multicapa vuelve a ser el peor de todos, probablemente por la escasez de datos para su entrenamiento, como ya hemos comentado anteriormente.

Cabe anotar, que, aunque el clasificador con el que hemos obtenido mejores resultados es con el de las máquinas de vectores de soporte, el resto de los clasificadores también han demostrado trabajar de forma bastante aceptable. Me gustaría destacar, sobre todo, la velocidad de reacción obtenida, y como los clasificadores son capaces de modificar su predicción según los cambios en la mano de una forma prácticamente instantánea.

7 CONCLUSIONES

7.1 Objetivos cumplidos

Si nos fijamos en los objetivos enumerados en la introducción, podemos ver que hemos cumplido con todos ellos de forma satisfactoria.

- En primer lugar, hemos sido capaces de comprender las bases del Machine Learning y de la visión por computador necesaria para la elaboración del proyecto.
- Se ha conseguido generar una base de datos bastante buena de forma muy sencilla, ya que el programa automáticamente filtra la imagen y busca en ella la mano para poder recortar el resto, y así solamente capturar la imagen correspondiente a la mano.
- Hemos conseguido, mediante distintos métodos, analizar cuáles pueden ser los mejores parámetros para nuestros clasificadores para así poder optimizar al máximo los algoritmos.
- Por lo general, hemos logrado que los 4 clasificadores sean capaces de realizar el trabajo que se les requería, ya que como mínimo, en el caso del perceptrón multicapa, hemos conseguido un porcentaje de acierto de casi un 90%.
- Se ha realizado una comparación entre estos cuatro métodos en la cual hemos podido elegir un claro ganador, las máquinas de vectores de soporte.
- Finalmente hemos conseguido trasladar nuestro algoritmo a una aplicación en tiempo real, la cual predice de forma totalmente instantánea, y además hemos podido aplicarla a un sencillo juego de pares y nones.

En resumen, podemos concluir con que hemos obtenido de forma satisfactoria el objetivo que teníamos al principio del trabajo. Se ha conseguido crear una aplicación que es capaz de predecir el número de dedos que está mostrando una mano en tiempo real, y de una forma totalmente fluida, sin la necesidad de utilizar guantes de colores ni ningún tipo de mecanismo más que la mano desnuda. Hemos visto además que métodos trabajan mejor sobre otros métodos, quedando como victorioso tanto en el análisis utilizando las imágenes de la base de datos, como en el análisis aplicado en la aplicación

en tiempo real, el método de Máquinas de Vectores de Soporte superando el 95% de acierto en el primer caso.

Nos ha sorprendido que el método del perceptrón multicapa haya sido el que peores resultados ha obtenido, ya que se postulaba como el método más potente de todos a priori. Queda demostrado que, en ciertos casos, los algoritmos de Machine Learning clásicos pueden trabajar incluso mejor que técnicas basadas en redes neuronales.

También hemos quedado bastante contentos con la forma en la que ha sido realizado el algoritmo, ya que al haber generado nosotros mismos nuestra propia base de datos, hemos creado una aplicación que parte absolutamente desde cero. Esto hace que la aplicación se extremadamente versátil. Simplemente cambiando los datos que hay en la base de datos, y ciertos parámetros de los clasificadores, podemos ser capaces de realizar una gran cantidad de tareas de aprendizaje sin modificar la estructura principal del programa. El mismo programa, por ejemplo, también podría servir para identificar distintos gestos realizados con la mano, como podrían ser los del lenguaje de signos.

7.2 Futuros trabajos y posibles mejoras

A continuación, detallamos futuras mejoras o aplicaciones que se podrían realizar siguiendo la misma línea del trabajo realizado.

- En primer lugar, la aplicación empezó con la idea de que pudiese ser aplicada a un robot humanoide, de forma que pudiésemos comunicarnos con él, o realizar algún tipo de juego como el de pares y nones estudiado, utilizando solamente las manos. En concreto se ha pensado en el robot humanoide TEO de la asociación RoboticsLab de la Universidad Carlos III de Madrid, ya que este robot también posee una cámara de luz estructurada. Con sus 28 grados de libertad, sería totalmente apto para llevar a cabo el juego de pares y nones.
- También existen ciertas mejoras que se podrían llevar a cabo para conseguir mejores resultados. El uso de una cámara de mayor calidad podría suponer un gran avance en la ejecución del programa. Asimismo, una ampliación de la base de datos podría suponer un mayor porcentaje de acierto y unos mejores

resultados en tiempo real, sobre todo si se aplica el método de Deep Learning del Perceptrón Multicapa.

- Por último, otro de los trabajos que nos quedan pendientes de realizar sería el de crear algún documento o tutorial en el que se explique todo el código de Python que se ha realizado de forma muy detallada para poder compartirlo con todo el departamento de automática de la universidad y que sea utilizado y mejorado por cualquier persona que esté realizando algún trabajo de la misma índole.

BIBLIOGRAFIA

[1] C. Raphael, «How Machine Learning Fuels Your Netflix Addiction,» *RT Insights*, p. Online, 5 Enero 2016.

[2] N. Capella, «Accenture automates 17,000 jobs without layoffs,» *The Stack*, p. Online, 19 Enero 2017.

[3] J. Lal Raheja, K. Das y A. Chaudhary, «Fingertip Detection: A Fast Method with Natural Hand,» *International Journal of Embedded Systems and Computer Engineering*, vol. 3, nº 2, pp. 85-88, 2011.

[4] M. Bhuyan, D. N. Raj y M. K. Kumar, «Fingertip Detection for Hand Pose Recognition,» *International Journal on Computer Science and Engineering (IJCSE)*, vol. 4, nº 3, pp. 501-511, 2012.

[5] G. Simion, C. David, V. Gui y C.-D. Căleanu, «Fingertip-based Real Time Tracking and Gesture Recognition for Natural User Interfaces,» *Acta Polytechnica Hungarica*, vol. 13, nº 5, pp. 189-204, 2016.

[6] W. Wu, C. Li, Z. Cheng, X. Zhang y L. Jin, «YOLSE: Egocentric Fingertip Detection from Single RGB Images,» de *IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017.

[7] I. Noor A., «Diverse Techniques for Geometric Features Extraction and Identification of Gesture Recognition,» *Computer Applications: An International Journal (CAIJ)*, vol. 3, nº 3, pp. 9-14, 2016.

[8] A. D. Yahya, M. N. Jan y J. Abdullah, «Static Hand Gestures: Fingertips Detection Based on Segmented Images,» *Journal of Computer Sciences*, vol. 11, nº 12, pp. 1090-1098, 2016.

[9] S. Michahial, N. R N, A. G N, B. H. Azeez, J. M R y R. K. Rani, «Hand gesture recognition using support vector machine,» *The International Journal Of Engineering And Science (IJES)*, vol. 4, nº 6, pp. 42-46, 2015.

[10] H. Rahman y J. Afrin, «Hand Gesture Recognition using Multiclass Support Vector Machine,» *International Journal of Computer Applications (0975 –*

8887), vol. 74, nº 1, pp. 39-43, 2013.

[11] J. Mackie y B. McCane, «Finger Detection with Decision Trees,» *University of Otago, Dept. of Computer Science.*

[12] P. Li, H. Ling, X. Li y C. Liao, «3D Hand Pose Estimation Using Randomized Decision Forest with Segmentation Index Points,» de *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[13] A. K. Nimbalkar, R. Karhe y C. Patil, «Face and Hand Gesture Recognition using Principle Component Analysis and kNN Classifier,» *International Journal of Computer Applications (0975 – 8887)*, vol. 99, nº 8, pp. 26-28, 2014.

[14] J. Vasquez, «GitHub,» 28 Julio 2017. [En línea]. Available: https://github.com/jv7/CNN-HowManyFingers. [Último acceso: 2 Septiembre 2018].

[15] P. Xu, «A Real-time Hand Gesture Recognition and Human-Computer Interaction System,» *Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities*, 2017.

[16] Y. LeCun, C. Cortes y C. J. Burges, «THE MNIST DATABASE of handwritten digits,» [En línea]. Available: http://yann.lecun.com/exdb/mnist/. [Último acceso: 2 Septiembre 2018].

[17] A. Dalmia, «Handwritten Digit Recognition using K Nearest Neighbour,» [En línea]. Available: https://researchweb.iiit.ac.in/~ayushi.dalmia/reports/Hand%20Digit%20Recognition.pdf. [Último acceso: 2 Septiembre 2018].

[18] S. Bernard, L. Heutte y S. Adam, «Using Random Forests for Handwritten Digit Recognition,» de *9th IAPR/IEEE International Conference on Document Analysis*, Curitiba, Brazil, 2007.

[19] E. Tuba, M. Tuba y D. Simian, «Handwritten Digit Recognition by Support Vector Machine Optimized by Bat Algorithm,» 2016.

[20] D. C. Ciresan, U. Meier, L. M. Gambardella y J. Schmidhuber, «Deep Big Multilayer Perceptrons For Digit Recognition,» 2012.

[21] L. Edell, «MUSINGS ON ML, DEEP LEARNING & AI,» 9 Diceimbre 2015. [En línea]. Available: https://scorecardstreet.wordpress.com/2015/12/09/is-machine-learning-the-new-epm-black/. [Último acceso: 3 Septiembre 2018].

[22] «Overview of Data Science,» 24 Noviembre 2016. [En línea]. Available: http://beta.cambridgespark.com/courses/jpm/01-module.html. [Último acceso: 3 Septiembre 2018].

[23] «Choosing the right estimator,» [En línea]. Available: http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html. [Último acceso: 3 Septiembre 2018].

[24] A. Ng, «Decision Trees: A Disastrous Tutorial,» Septiembre 2016. [En línea]. Available: https://www.kdnuggets.com/2016/09/decision-trees-disastrous-overview.html. [Último acceso: 3 Septiembre 2018].

[25] A. Manglick, «Arun Manglick - Artificial Intelligence & Machine/Deep Learning - Decision Tree : Classification,» 9 Julio 2017. [En línea]. Available: http://arun-aiml.blogspot.com/2017/07/decision-tree-classification.html. [Último acceso: 3 Septiembre 2018].

[26] C. Gini, Variabilità e mutabilità, Salvemini, 1912.

[27] C. Cortes y V. Vapnik, «Support-Vector Networks,» *Machine Learning*, nº 20, pp. 273-297, 1995.

[28] P. Goyal, «How can I use a Support Vector Machine in regression tasks (SVM)?,» 29 Mayo 2017. [En línea]. Available: https://www.quora.com/How-can-I-use-a-Support-Vector-Machine-in-regression-tasks-SVM. [Último acceso: 3 Septiembre 2018].

[29] «Java crumbs Short remarks froma Java world - Machine learning for dummies – Support Vector Machines,» 7 Febrero 2016. [En línea]. Available: https://blog.krecan.net/2016/02/07/machine-learning-for-dummies-support-vector-machines/. [Último acceso: 3 Septiembre 2018].

[30] W. F, «Github,» 24 Septiembre 2016. [En línea]. Available: https://gist.github.com/WittmannF/60680723ed8dd0cb993051a7448f7805.

[Último acceso: 3 Septimebre 2018].

[31] S. Alaliyat, *Video - based Fall Detection in Elderly's Houses,* Department of Computer Science and Media Technology Gjøvik University College, 2008.

[32] Ravarani, «Misleading modelling: overfitting, cross-validation, and the bias-variance trade-off,» 24 Marzo 2016. [En línea]. Available: https://cambridgecoding.wordpress.com/2016/03/24/misleading-modelling-overfitting-cross-validation-and-the-bias-variance-trade-off/. [Último acceso: 3 Septiembre 2018].

[33] M. A. Nielsen, «Neural Networks and Deep Learning,» 2 Diciembre 2017. [En línea]. Available: http://neuralnetworksanddeeplearning.com/chap6.html. [Último acceso: 3 Septiembre 2018].

[34] U. Karn, «A Quick Introduction to Neural Networks,» Noviembre 2016. [En línea]. Available: https://www.kdnuggets.com/2016/11/quick-introduction-neural-networks.html. [Último acceso: 3 Septiembre 2018].

[35] L. G. Roberts, «Machine perception of three-dimensional solids,» 1963.

[36] J. Park, C. Kim, J. Na, J. Yi y M. Turk, «Using structured light for efficient depth edge detection,» *Image and Vision Computing 26,* pp. 1449-1465, 2008.

[37] F. W. DePiero y M. M. Trivedi, «3-D Computer Vision Using Structured Light: Design, Calibration and Implementation Issues,» *Computer Vision and Robotics Research Laboratory Electrical and Computer Engineering Department The University of Tennessee, Knoxville.*

[38] A. M. C. Araujo, «TANGO WITH CODE A blog about frustration and anger,» 15 Noviembre 2015. [En línea]. Available: https://picoledelimao.github.io/blog/2015/11/15/fingertip-detection-on-opencv/. [Último acceso: 1 Septiembre 2018].

[39] Z.-h. Chen, J.-T. Kim, J. Liang, J. Zhang y Y.-B. Yuan, «Real-Time Hand Gesture Recognition Using Finger Segmentation,» *ScientificWorldJournal,* 2014.

[40] A. S. Ray, «Color gamut transform pairs,» *'78 Proceedings of the 5th annual conference on Computer graphics and interactive techniques,* pp. 12-19, 1978.

[41] J. Buhagiar, *Automatic Segmentation of Indoor and Outdoor scenes from Visual Lifelogging*, Faculty of ICT - University of Malta, 2017.

[42] C. J. C. Burgues, «A Tutorial on Support Vector Machines for Pattern Recognition,» *Data Mining and Knowledge Discovery,* nº 2, pp. 121-167, 1998.

[43] P. W. -. M. OpenCourseWare, «16. Learning: Support Vector Machines,» YouTube, 10 Enero 2014. [En línea]. Available: https://www.youtube.com/watch?v=_PwhiWxHK8o. [Último acceso: 3 Septiembre 2018].

[44] G. Bradsky y A. Kaehler, Learning OpenCV, O'Reilly, 2008.

[45] «Opencv - Open Source Computer Vision,» 23 Diciembre 2016. [En línea]. Available: https://docs.opencv.org/3.2.0/index.html. [Último acceso: 3 Septiembre 2018].

[46] J. D. Hunter, «Matplotlib: A 2D graphics environment,» *Computing In Science \& Engineering*, vol. 9, nº 3, pp. 90-95, 2007.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot y E. Duchesnay, «Scikit-learn: Machine Learning in Python,» *Journal of Machine Learning Research,* vol. 12, pp. 2825-28330, 2011.

[48] W. S. Percybrooks, «Inteligencia Artificial: Impacto socio-económico,» [En línea]. Available: https://www.uninorte.edu.co/documents/71530/13179136/Percybrooks+VII+Jornada.pdf/44bb232c-9224-4095-8c2f-943013444388. [Último acceso: 1 Septiembre 2018].

[49] Y. Zhang, «3D Machine Learning,» 2018. [En línea]. Available: https://github.com/timzhang642/3D-Machine-Learning. [Último acceso: 7 Septiembre 2018].

[50] CS50, «Computer Vision and Machine Learning, by Nick Wong,» 31 Octubre 2017. [En línea]. Available:

https://www.youtube.com/watch?v=PMsY5PlDVQw. [Último acceso: 7 Septiembre 2018].

[51] W. Labs, «Learning To See,» 15 Abril 2016. [En línea]. Available: https://www.youtube.com/watch?v=i8D90DkCLhI. [Último acceso: 7 Septiembre 2018].

[52] K.-C. Lu y D.-L. Yang, «Image Processing and Image Mining using Decision Trees,» *Journal of Information Science and Engineerig,* vol. 25, pp. 989-1003, 2009.

[53] D. Korbut, «Machine Learning Algorithms: Which One to Choose for Your Problem,» 26 Octubre 2017. [En línea]. Available: https://blog.statsbot.co/machine-learning-algorithms-183cc73197c. [Último acceso: 7 Septiembre 2018].

[54] sendtex, «Thresholding - OpenCV with Python for Image and Video Analysis,» 22 Diciembre 2015. [En línea]. Available: https://blog.statsbot.co/machine-learning-algorithms-183cc73197c. [Último acceso: 7 Septiembre 2018].

[55] Sreehari, «Weekend project: sign language and static-gesture recognition using sci-kit learn,» 26 Diciembre 2016. [En línea]. Available: https://medium.freecodecamp.org/weekend-projects-sign-language-and-static-gesture-recognition-using-scikit-learn-60813d600e79. [Último acceso: 7 Septiembre 2018].

[56] M. Lees, «How to create your own dataset for machine learning,» 2017. [En línea]. Available: https://steemit.com/technology/@howo/how-to-create-your-own-dataset-for-machine-learning. [Último acceso: 7 Septiembre 2018].

[57] A. Mukherjee, «HandGesturePy,» 2016. [En línea]. Available: https://github.com/arijitx/HandGesturePy. [Último acceso: 7 Septiembre 2018].

[58] PyRevolution, «Handwriting Recognition Python,» 11 Diciembre 2015. [En línea]. Available: https://www.youtube.com/watch?v=PO4hePKWIGQ. [Último acceso: 7 Septiembre 2018].

[59] S. Rsachka, «Neuronal Network - Multilayer Perceptron,» 2017. [En línea].

Available: https://rasbt.github.io/mlxtend/user_guide/classifier/MultiLayerPerceptron/. [Último acceso: 7 Septiembre 2018].

[60] M. Sunasra, «Performance Metrics for Classification problems in Machine Learning,» 11 Noviembre 2017. [En línea]. Available: https://medium.com/greyatom/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b. [Último acceso: 11 Septiembre 2018].

[61] T. Hastie, R. Tibshirani y J. Friedman, The Elements os Statistical Learning - Data Mining, Inference and Prediction, Springer, 2009.

[62] L. A. González, «Modelos de Clasificación basados en Máquinas de Vectores de Soporte,» *Departamento de Economía Aplicada - Universidad de Sevilla.*

[63] L. Wang, Support Vector Machines: Theory and Applications, Berlin: Springer, 2005.

[64] M. OpenCourseWeb, «Learning: Support Vector Machines,» 10 Enero 2014. [En línea]. Available: https://www.youtube.com/watch?v=_PwhiWxHK8o. [Último acceso: 7 Septiembre 2018].

[65] T. Fletcher, «Support Vector Machines Explained,» *Univesity College de Londres,* 2008.

[66] A. Smola y S. Vishwanathan, Introduction to Machine Learning, Cambridge University Press, 2008.

[67] A. M. González, F. J. A. Martínez de Pisón, A. V. E. Pernía, F. E. Alba, M. L. Castejón, J. M. Ordieres y E. G. Vergara, Tecnicas y algoritmos básicos de visión artificial, Universidad de la Rioja Servicio de Publicaciones, 2006.

[68] L. Roberts, «Machine perception of 3D solids,» *Optical and Electro-Optical Information Processing*, pp. 159-197, 1965.

ANEXO

En el Anexo hemos incluido todo el código de Python utilizado en la elaboración del proyecto. Este proyecto ha sido desarrollado con la aplicación web Jupyter Notebook, de la cual hemos querido mantener su formato y estructura original. Cada apartado del código se pude compilar de forma independiente del resto.

Crear Base de Datos

In [1]:

```
#Ejecutar esta parte solo cuando no se tenga una base de datos creada

import cv2
import os
import numpy as np
import matplotlib.pyplot as plt

"""

Estructura de carpetas necesaria para guardar las imagenes

dataset
    |
    |----------1_finger
    |       .   |-------hand_0.png
    |       .   |        .
    |       .   |        .
    |       .   |        .
    |       .   |-------hand_n.png
    |       .
    |----------5_fingers

"""

#variables para la captura de video
cap_rbg = cv2.VideoCapture(0)
cap_depth = cv2.VideoCapture(1)

#seleccionamos el tamaño que queramos qeu tengan las imágenes
im_size = 28

#contamos número de archivos que ya hay en la carpeta
dir_name='dataset_28/1_finger'
numfiles = len([f for f in os.listdir(dir_name)
                if os.path.isfile(os.path.join(dir_name, f))])

#inicializamos cuadrado de la roi
x,y,w,h = 0,0,100,100
```

```
while True:

    #guardamos el vídeo en una variable
    ret, frame_rgb = cap_rbg.read()
    ret, frame_depth = cap_depth.read()

    #aplicamos efecto espejo
    frame_depth = cv2.flip(frame_depth,1)

    #seleccionamos el color del fondo para eliminarlo
    lower_green = np.array([0, 154, 0])
    upper_green = np.array([0, 154, 0])
    mask = cv2.inRange(frame_depth, lower_green, upper_green)

    #aplicamos un filtro para eliminar ruido
    median = cv2.medianBlur(mask ,15)

    #Invierto colores de la imagen para que funcione bien la captura de contorno
    median = 255 - median;

    #reducimos el tamaño de la imagen
    resized_image = cv2.resize(median, (200, 150))

    #Buscamos el contorno en la imagen filtrada
    median, contours, hierarchy = cv2.findContours(median, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) != 0:
        # Dibujamos los contornos
        cv2.drawContours(median, contours, -1, (122,122,0), 3)

        #Buscamos el area maxima
        c = max(contours, key = cv2.contourArea)

        #Declaramos las medidas del rectangulo de contorno maximo
        x,y,w,h = cv2.boundingRect(c)

        #Dibujamos el contorno
        cv2.rectangle(median,(x,y),(x+w,y+h),(255,255,255),2)

    roi_max = median[y:y+h,x:x+w]

    resized_roi = cv2.resize(roi_max, (im_size, im_size))

    #mostramos el vídeo por pantalla
    cv2.imshow('frame_depth', frame_depth)
    cv2.imshow('median', median)
    cv2.imshow('roi_max', roi_max)

    #guardamos los frames de la mano en formato png al pulsar la tecla C
    if cv2.waitKey(1) & 0xFF == ord('c'):
        name = "hand%d.png"%(numfiles + 1)
```

```
        cv2.imwrite(os.path.join(dir_name, name), resized_roi)
        numfiles += 1

    #salimos de la ejecucion pulsando Q
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap_rbg.release()
cap_depth.release()
cv2.destroyAllWindows()
cv2.waitKey(0)
```

Out [1]:

```
-1
```

Importar los Datos

In [2]:

```
"""
Ejecutar esta parte la primera vez que queramos
ejecutar alguna de las demás partes siguientes.

En esta parte se declaran todas las librerías que se vayan a utilizar
y se importan los datos de la base de datos.
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn import ensemble
from sklearn import svm
from sklearn.cross_validation import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn import neighbors
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from os import listdir
from glob import glob
from IPython.display import Audio, display
import cv2
from time import time
import seaborn as sn
import pandas as pd

#Cargamos todas imágenes de la base de datos

%matplotlib inline

path = "dataset_28"
resize = True
percentage = 100
numb_label = []

classes = listdir(path)

image_list = []
labels = []
for classe in classes:
    for filename in glob(path+'/'+classe+'/*.png'):

        image_list.append(cv2.cvtColor(cv2.imread(filename), cv2.COLOR_BGR2GRAY))

        label = np.zeros(len(classes))
        label[classes.index(classe)]=1
        labels.append(label)

indice = np.random.permutation(len(image_list))[:int(len(image_list)*percentage/100)]
```

```
#guardamos los datos en la variable images
images = np.array([image_list[x] for x in indice])
#guardamos las etiquetas en la variable labels
labels = np.array([labels[x] for x in indice])
classes = np.array(classes)

#redimensionamos las imágenes
nsamples, nx, ny = images.shape
data = images.reshape((nsamples,nx*ny))

#numb_label = etiquetas numericas
for i in range (0,len(image_list)):
    numb_label.append(labels[i].argmax())
numb_label=np.array(numb_label)

#ordenamos las etiquetas
for i in range (0,len(image_list)):
    if numb_label[i]==0: numb_label[i]=5
    elif numb_label[i]==1: numb_label[i]=1
    elif numb_label[i]==2: numb_label[i]=4
    elif numb_label[i]==3: numb_label[i]=3
    elif numb_label[i]==4: numb_label[i]=2


print "Working with {} images\n".format(len(image_list))

ratio = len(image_list)*0.8
ratio = int(ratio)
print "Training images: ",ratio
print "Test images: ",len(image_list)-ratio
print ""
```

Out [2]:

```
Working with 1574 images

Training images:  1259
Test images:  315
```

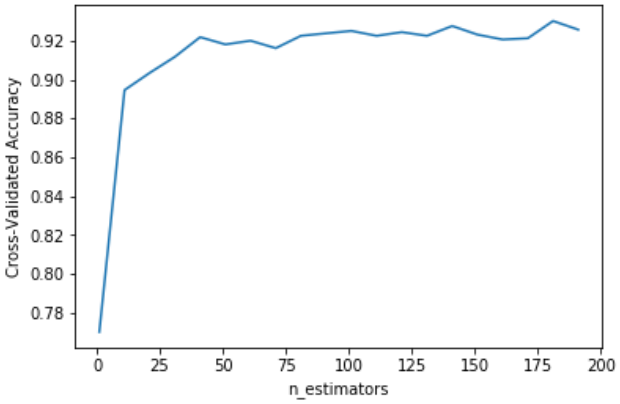
Ver graficas de valores concretos para parametrización

In [3]:

```
#en este apartado representamos una variable frente a la precision

parameter = list(xrange(1, 200, 10))
k_scores = []
for k in parameter:
    clf = ensemble.RandomForestClassifier(n_estimators = k)
    scores = cross_val_score(clf, data, numb_label, cv=5, scoring='accuracy')
```

```
    k_scores.append(scores.mean())

# imprimimos
plt.plot(parameter, k_scores)
plt.xlabel('n_estimators')
plt.ylabel('Cross-Validated Accuracy')
```

Out [3]:



Grid Search

Random Forest

In[4]:

```
#declaramos los parámetros que vamos a utilizar

parameters_rf = {'n_estimators':[25,30,35,40,45,50,55,60,65,75],
                 'criterion':["gini", "entropy"],
                 'max_depth':[5,6,7,8,9,10,11,12]}

clf_rf = ensemble.RandomForestClassifier()
tunning_rf = GridSearchCV(clf_rf, parameters_rf, cv = 5, scoring = 'accuracy')
tunning_rf.fit(data, numb_label)

print "Best parameters set found on development set:"
print tunning_rf.best_params_

print ""

print "With score:"
print tunning_rf.best_score_

print ""

print "Total Scores:"
print ""
means = tunning_rf.cv_results_['mean_test_score']
```

```
stds = tunning_rf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, tunning_rf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

Out [4]:

```
Best parameters set found on development set:
{'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 12}

With score:
0.932020330368488

Total Scores:

0.825 (+/-0.035) for {'n_estimators': 25, 'criterion': 'gini', 'max_depth': 5}
0.816 (+/-0.028) for {'n_estimators': 30, 'criterion': 'gini', 'max_depth': 5}
0.829 (+/-0.043) for {'n_estimators': 35, 'criterion': 'gini', 'max_depth': 5}
0.830 (+/-0.050) for {'n_estimators': 40, 'criterion': 'gini', 'max_depth': 5}
0.820 (+/-0.024) for {'n_estimators': 45, 'criterion': 'gini', 'max_depth': 5}
0.824 (+/-0.044) for {'n_estimators': 50, 'criterion': 'gini', 'max_depth': 5}
0.832 (+/-0.042) for {'n_estimators': 55, 'criterion': 'gini', 'max_depth': 5}
0.834 (+/-0.051) for {'n_estimators': 60, 'criterion': 'gini', 'max_depth': 5}
0.829 (+/-0.040) for {'n_estimators': 65, 'criterion': 'gini', 'max_depth': 5}
0.831 (+/-0.027) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 5}
0.832 (+/-0.057) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 5}
0.844 (+/-0.042) for {'n_estimators': 25, 'criterion': 'gini', 'max_depth': 6}
0.853 (+/-0.025) for {'n_estimators': 30, 'criterion': 'gini', 'max_depth': 6}
0.855 (+/-0.040) for {'n_estimators': 35, 'criterion': 'gini', 'max_depth': 6}
0.868 (+/-0.044) for {'n_estimators': 40, 'criterion': 'gini', 'max_depth': 6}
0.860 (+/-0.031) for {'n_estimators': 45, 'criterion': 'gini', 'max_depth': 6}
0.863 (+/-0.046) for {'n_estimators': 50, 'criterion': 'gini', 'max_depth': 6}
0.859 (+/-0.026) for {'n_estimators': 55, 'criterion': 'gini', 'max_depth': 6}
0.859 (+/-0.019) for {'n_estimators': 60, 'criterion': 'gini', 'max_depth': 6}
0.856 (+/-0.026) for {'n_estimators': 65, 'criterion': 'gini', 'max_depth': 6}
0.860 (+/-0.034) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 6}
0.856 (+/-0.027) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 6}
0.874 (+/-0.027) for {'n_estimators': 25, 'criterion': 'gini', 'max_depth': 7}
0.886 (+/-0.041) for {'n_estimators': 30, 'criterion': 'gini', 'max_depth': 7}
0.879 (+/-0.004) for {'n_estimators': 35, 'criterion': 'gini', 'max_depth': 7}
0.870 (+/-0.010) for {'n_estimators': 40, 'criterion': 'gini', 'max_depth': 7}
0.881 (+/-0.032) for {'n_estimators': 45, 'criterion': 'gini', 'max_depth': 7}
0.890 (+/-0.033) for {'n_estimators': 50, 'criterion': 'gini', 'max_depth': 7}
0.881 (+/-0.023) for {'n_estimators': 55, 'criterion': 'gini', 'max_depth': 7}
0.879 (+/-0.017) for {'n_estimators': 60, 'criterion': 'gini', 'max_depth': 7}
0.883 (+/-0.026) for {'n_estimators': 65, 'criterion': 'gini', 'max_depth': 7}
0.882 (+/-0.031) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 7}
0.886 (+/-0.023) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 7}
0.891 (+/-0.036) for {'n_estimators': 25, 'criterion': 'gini', 'max_depth': 8}
0.900 (+/-0.032) for {'n_estimators': 30, 'criterion': 'gini', 'max_depth': 8}
0.902 (+/-0.016) for {'n_estimators': 35, 'criterion': 'gini', 'max_depth': 8}
0.893 (+/-0.021) for {'n_estimators': 40, 'criterion': 'gini', 'max_depth': 8}
0.900 (+/-0.010) for {'n_estimators': 45, 'criterion': 'gini', 'max_depth': 8}
0.900 (+/-0.020) for {'n_estimators': 50, 'criterion': 'gini', 'max_depth': 8}
0.895 (+/-0.024) for {'n_estimators': 55, 'criterion': 'gini', 'max_depth': 8}
0.904 (+/-0.029) for {'n_estimators': 60, 'criterion': 'gini', 'max_depth': 8}
0.896 (+/-0.025) for {'n_estimators': 65, 'criterion': 'gini', 'max_depth': 8}
0.904 (+/-0.034) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 8}
0.904 (+/-0.023) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 8}
0.902 (+/-0.018) for {'n_estimators': 25, 'criterion': 'gini', 'max_depth': 9}
0.902 (+/-0.023) for {'n_estimators': 30, 'criterion': 'gini', 'max_depth': 9}
0.900 (+/-0.016) for {'n_estimators': 35, 'criterion': 'gini', 'max_depth': 9}
0.903 (+/-0.021) for {'n_estimators': 40, 'criterion': 'gini', 'max_depth': 9}
0.907 (+/-0.013) for {'n_estimators': 45, 'criterion': 'gini', 'max_depth': 9}
0.912 (+/-0.024) for {'n_estimators': 50, 'criterion': 'gini', 'max_depth': 9}
0.912 (+/-0.007) for {'n_estimators': 55, 'criterion': 'gini', 'max_depth': 9}
0.914 (+/-0.011) for {'n_estimators': 60, 'criterion': 'gini', 'max_depth': 9}
0.912 (+/-0.014) for {'n_estimators': 65, 'criterion': 'gini', 'max_depth': 9}
0.909 (+/-0.016) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 9}
```

```
0.908 (+/-0.019) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 9}
0.909 (+/-0.014) for {'n_estimators': 25, 'criterion': 'gini', 'max_depth': 10}
0.913 (+/-0.017) for {'n_estimators': 30, 'criterion': 'gini', 'max_depth': 10}
0.916 (+/-0.011) for {'n_estimators': 35, 'criterion': 'gini', 'max_depth': 10}
0.915 (+/-0.020) for {'n_estimators': 40, 'criterion': 'gini', 'max_depth': 10}
0.912 (+/-0.027) for {'n_estimators': 45, 'criterion': 'gini', 'max_depth': 10}
0.922 (+/-0.032) for {'n_estimators': 50, 'criterion': 'gini', 'max_depth': 10}
0.915 (+/-0.036) for {'n_estimators': 55, 'criterion': 'gini', 'max_depth': 10}
0.916 (+/-0.018) for {'n_estimators': 60, 'criterion': 'gini', 'max_depth': 10}
0.921 (+/-0.012) for {'n_estimators': 65, 'criterion': 'gini', 'max_depth': 10}
0.922 (+/-0.016) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 10}
0.925 (+/-0.014) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 10}
0.898 (+/-0.013) for {'n_estimators': 25, 'criterion': 'gini', 'max_depth': 11}
0.919 (+/-0.011) for {'n_estimators': 30, 'criterion': 'gini', 'max_depth': 11}
0.917 (+/-0.015) for {'n_estimators': 35, 'criterion': 'gini', 'max_depth': 11}
0.916 (+/-0.020) for {'n_estimators': 40, 'criterion': 'gini', 'max_depth': 11}
0.916 (+/-0.011) for {'n_estimators': 45, 'criterion': 'gini', 'max_depth': 11}
0.914 (+/-0.018) for {'n_estimators': 50, 'criterion': 'gini', 'max_depth': 11}
0.921 (+/-0.019) for {'n_estimators': 55, 'criterion': 'gini', 'max_depth': 11}
0.929 (+/-0.009) for {'n_estimators': 60, 'criterion': 'gini', 'max_depth': 11}
0.920 (+/-0.016) for {'n_estimators': 65, 'criterion': 'gini', 'max_depth': 11}
0.924 (+/-0.011) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 11}
0.921 (+/-0.020) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 11}
0.915 (+/-0.024) for {'n_estimators': 25, 'criterion': 'gini', 'max_depth': 12}
0.913 (+/-0.019) for {'n_estimators': 30, 'criterion': 'gini', 'max_depth': 12}
0.910 (+/-0.018) for {'n_estimators': 35, 'criterion': 'gini', 'max_depth': 12}
0.917 (+/-0.015) for {'n_estimators': 40, 'criterion': 'gini', 'max_depth': 12}
0.919 (+/-0.016) for {'n_estimators': 45, 'criterion': 'gini', 'max_depth': 12}
0.924 (+/-0.012) for {'n_estimators': 50, 'criterion': 'gini', 'max_depth': 12}
0.916 (+/-0.023) for {'n_estimators': 55, 'criterion': 'gini', 'max_depth': 12}
0.916 (+/-0.023) for {'n_estimators': 60, 'criterion': 'gini', 'max_depth': 12}
0.925 (+/-0.017) for {'n_estimators': 65, 'criterion': 'gini', 'max_depth': 12}
0.925 (+/-0.017) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 12}
0.922 (+/-0.002) for {'n_estimators': 75, 'criterion': 'gini', 'max_depth': 12}
0.815 (+/-0.040) for {'n_estimators': 25, 'criterion': 'entropy', 'max_depth': 5}
0.830 (+/-0.053) for {'n_estimators': 30, 'criterion': 'entropy', 'max_depth': 5}
0.837 (+/-0.047) for {'n_estimators': 35, 'criterion': 'entropy', 'max_depth': 5}
0.831 (+/-0.038) for {'n_estimators': 40, 'criterion': 'entropy', 'max_depth': 5}
0.832 (+/-0.028) for {'n_estimators': 45, 'criterion': 'entropy', 'max_depth': 5}
0.840 (+/-0.045) for {'n_estimators': 50, 'criterion': 'entropy', 'max_depth': 5}
0.821 (+/-0.049) for {'n_estimators': 55, 'criterion': 'entropy', 'max_depth': 5}
0.841 (+/-0.036) for {'n_estimators': 60, 'criterion': 'entropy', 'max_depth': 5}
0.836 (+/-0.044) for {'n_estimators': 65, 'criterion': 'entropy', 'max_depth': 5}
0.833 (+/-0.048) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 5}
0.840 (+/-0.041) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 5}
0.855 (+/-0.029) for {'n_estimators': 25, 'criterion': 'entropy', 'max_depth': 6}
0.855 (+/-0.024) for {'n_estimators': 30, 'criterion': 'entropy', 'max_depth': 6}
0.865 (+/-0.018) for {'n_estimators': 35, 'criterion': 'entropy', 'max_depth': 6}
0.863 (+/-0.030) for {'n_estimators': 40, 'criterion': 'entropy', 'max_depth': 6}
0.870 (+/-0.046) for {'n_estimators': 45, 'criterion': 'entropy', 'max_depth': 6}
0.869 (+/-0.034) for {'n_estimators': 50, 'criterion': 'entropy', 'max_depth': 6}
0.867 (+/-0.019) for {'n_estimators': 55, 'criterion': 'entropy', 'max_depth': 6}
0.871 (+/-0.011) for {'n_estimators': 60, 'criterion': 'entropy', 'max_depth': 6}
0.871 (+/-0.020) for {'n_estimators': 65, 'criterion': 'entropy', 'max_depth': 6}
0.871 (+/-0.027) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 6}
0.870 (+/-0.040) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 6}
0.877 (+/-0.012) for {'n_estimators': 25, 'criterion': 'entropy', 'max_depth': 7}
0.885 (+/-0.020) for {'n_estimators': 30, 'criterion': 'entropy', 'max_depth': 7}
0.893 (+/-0.035) for {'n_estimators': 35, 'criterion': 'entropy', 'max_depth': 7}
0.892 (+/-0.022) for {'n_estimators': 40, 'criterion': 'entropy', 'max_depth': 7}
0.889 (+/-0.038) for {'n_estimators': 45, 'criterion': 'entropy', 'max_depth': 7}
0.889 (+/-0.023) for {'n_estimators': 50, 'criterion': 'entropy', 'max_depth': 7}
0.889 (+/-0.008) for {'n_estimators': 55, 'criterion': 'entropy', 'max_depth': 7}
0.894 (+/-0.022) for {'n_estimators': 60, 'criterion': 'entropy', 'max_depth': 7}
0.895 (+/-0.011) for {'n_estimators': 65, 'criterion': 'entropy', 'max_depth': 7}
0.898 (+/-0.022) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 7}
0.895 (+/-0.030) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 7}
0.904 (+/-0.026) for {'n_estimators': 25, 'criterion': 'entropy', 'max_depth': 8}
0.911 (+/-0.028) for {'n_estimators': 30, 'criterion': 'entropy', 'max_depth': 8}
0.905 (+/-0.036) for {'n_estimators': 35, 'criterion': 'entropy', 'max_depth': 8}
0.905 (+/-0.026) for {'n_estimators': 40, 'criterion': 'entropy', 'max_depth': 8}
```

```
0.912 (+/-0.024) for {'n_estimators': 45, 'criterion': 'entropy', 'max_depth': 8}
0.914 (+/-0.010) for {'n_estimators': 50, 'criterion': 'entropy', 'max_depth': 8}
0.912 (+/-0.016) for {'n_estimators': 55, 'criterion': 'entropy', 'max_depth': 8}
0.909 (+/-0.015) for {'n_estimators': 60, 'criterion': 'entropy', 'max_depth': 8}
0.908 (+/-0.020) for {'n_estimators': 65, 'criterion': 'entropy', 'max_depth': 8}
0.914 (+/-0.012) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 8}
0.910 (+/-0.023) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 8}
0.907 (+/-0.020) for {'n_estimators': 25, 'criterion': 'entropy', 'max_depth': 9}
0.904 (+/-0.012) for {'n_estimators': 30, 'criterion': 'entropy', 'max_depth': 9}
0.907 (+/-0.019) for {'n_estimators': 35, 'criterion': 'entropy', 'max_depth': 9}
0.914 (+/-0.014) for {'n_estimators': 40, 'criterion': 'entropy', 'max_depth': 9}
0.916 (+/-0.031) for {'n_estimators': 45, 'criterion': 'entropy', 'max_depth': 9}
0.919 (+/-0.016) for {'n_estimators': 50, 'criterion': 'entropy', 'max_depth': 9}
0.917 (+/-0.023) for {'n_estimators': 55, 'criterion': 'entropy', 'max_depth': 9}
0.918 (+/-0.016) for {'n_estimators': 60, 'criterion': 'entropy', 'max_depth': 9}
0.919 (+/-0.021) for {'n_estimators': 65, 'criterion': 'entropy', 'max_depth': 9}
0.928 (+/-0.021) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 9}
0.922 (+/-0.017) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 9}
0.915 (+/-0.021) for {'n_estimators': 25, 'criterion': 'entropy', 'max_depth': 10}
0.917 (+/-0.014) for {'n_estimators': 30, 'criterion': 'entropy', 'max_depth': 10}
0.916 (+/-0.025) for {'n_estimators': 35, 'criterion': 'entropy', 'max_depth': 10}
0.922 (+/-0.011) for {'n_estimators': 40, 'criterion': 'entropy', 'max_depth': 10}
0.914 (+/-0.020) for {'n_estimators': 45, 'criterion': 'entropy', 'max_depth': 10}
0.921 (+/-0.011) for {'n_estimators': 50, 'criterion': 'entropy', 'max_depth': 10}
0.918 (+/-0.009) for {'n_estimators': 55, 'criterion': 'entropy', 'max_depth': 10}
0.920 (+/-0.014) for {'n_estimators': 60, 'criterion': 'entropy', 'max_depth': 10}
0.929 (+/-0.006) for {'n_estimators': 65, 'criterion': 'entropy', 'max_depth': 10}
0.925 (+/-0.009) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 10}
0.922 (+/-0.021) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 10}
0.913 (+/-0.023) for {'n_estimators': 25, 'criterion': 'entropy', 'max_depth': 11}
0.916 (+/-0.024) for {'n_estimators': 30, 'criterion': 'entropy', 'max_depth': 11}
0.921 (+/-0.026) for {'n_estimators': 35, 'criterion': 'entropy', 'max_depth': 11}
0.924 (+/-0.016) for {'n_estimators': 40, 'criterion': 'entropy', 'max_depth': 11}
0.924 (+/-0.014) for {'n_estimators': 45, 'criterion': 'entropy', 'max_depth': 11}
0.921 (+/-0.025) for {'n_estimators': 50, 'criterion': 'entropy', 'max_depth': 11}
0.924 (+/-0.013) for {'n_estimators': 55, 'criterion': 'entropy', 'max_depth': 11}
0.927 (+/-0.006) for {'n_estimators': 60, 'criterion': 'entropy', 'max_depth': 11}
0.926 (+/-0.013) for {'n_estimators': 65, 'criterion': 'entropy', 'max_depth': 11}
0.924 (+/-0.012) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 11}
0.926 (+/-0.031) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 11}
0.916 (+/-0.010) for {'n_estimators': 25, 'criterion': 'entropy', 'max_depth': 12}
0.917 (+/-0.021) for {'n_estimators': 30, 'criterion': 'entropy', 'max_depth': 12}
0.925 (+/-0.008) for {'n_estimators': 35, 'criterion': 'entropy', 'max_depth': 12}
0.919 (+/-0.025) for {'n_estimators': 40, 'criterion': 'entropy', 'max_depth': 12}
0.921 (+/-0.011) for {'n_estimators': 45, 'criterion': 'entropy', 'max_depth': 12}
0.924 (+/-0.018) for {'n_estimators': 50, 'criterion': 'entropy', 'max_depth': 12}
0.925 (+/-0.014) for {'n_estimators': 55, 'criterion': 'entropy', 'max_depth': 12}
0.929 (+/-0.014) for {'n_estimators': 60, 'criterion': 'entropy', 'max_depth': 12}
0.931 (+/-0.013) for {'n_estimators': 65, 'criterion': 'entropy', 'max_depth': 12}
0.928 (+/-0.022) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 12}
0.932 (+/-0.005) for {'n_estimators': 75, 'criterion': 'entropy', 'max_depth': 12}
```

Support Vector Machines

In [5]:

```
#declaramos los parámetros que vamos a utilizar

parameters_svm = {'kernel': ['poly'],'C': [1e-17,1e-15,1e-12,1e-8] ,
                  'gamma': [1e-4,1e-3,1e-2],
                  'degree': [1,2,3,4,5,6]}

clf_svm = svm.SVC()
tunning_svm = GridSearchCV(clf_svm, parameters_svm, cv = 5)
```

```
tunning_svm.fit(data, numb_label)

print "Best parameters set found on development set:"
print tunning_svm.best_params_

print ""

print "With score:"
print tunning_svm.best_score_

print ""

print "Total Scores:"
print ""
means = tunning_svm.cv_results_['mean_test_score']
stds = tunning_svm.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, tunning_svm.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

Out [5]:

```
Best parameters set found on development set:
{'kernel': 'poly', 'C': 1e-12, 'gamma': 0.01, 'degree': 3}

With score:
0.9504447268106735

Total Scores:

0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.0001, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.001, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.01, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.0001, 'degree': 2}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.001, 'degree': 2}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.01, 'degree': 2}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.0001, 'degree': 3}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.001, 'degree': 3}
0.630 (+/-0.005) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.01, 'degree': 3}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.0001, 'degree': 4}
0.913 (+/-0.011) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.001, 'degree': 4}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.01, 'degree': 4}
0.813 (+/-0.033) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.0001, 'degree': 5}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.001, 'degree': 5}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.01, 'degree': 5}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.0001, 'degree': 6}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.001, 'degree': 6}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-17, 'gamma': 0.01, 'degree': 6}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.0001, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.001, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.01, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.0001, 'degree': 2}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.001, 'degree': 2}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.01, 'degree': 2}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.0001, 'degree': 3}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.001, 'degree': 3}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.01, 'degree': 3}
0.388 (+/-0.010) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.0001, 'degree': 4}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.001, 'degree': 4}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.01, 'degree': 4}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.0001, 'degree': 5}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.001, 'degree': 5}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.01, 'degree': 5}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.0001, 'degree': 6}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.001, 'degree': 6}
```

```
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-15, 'gamma': 0.01, 'degree': 6}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.0001, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.001, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.01, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.0001, 'degree': 2}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.001, 'degree': 2}
0.418 (+/-0.009) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.01, 'degree': 2}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.0001, 'degree': 3}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.001, 'degree': 3}
0.950 (+/-0.019) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.01, 'degree': 3}
0.950 (+/-0.019) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.0001, 'degree': 4}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.001, 'degree': 4}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.01, 'degree': 4}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.0001, 'degree': 5}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.001, 'degree': 5}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.01, 'degree': 5}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.0001, 'degree': 6}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.001, 'degree': 6}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-12, 'gamma': 0.01, 'degree': 6}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.0001, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.001, 'degree': 1}
0.234 (+/-0.001) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.01, 'degree': 1}
0.418 (+/-0.009) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.0001, 'degree': 2}
0.915 (+/-0.015) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.001, 'degree': 2}
0.942 (+/-0.019) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.01, 'degree': 2}
0.950 (+/-0.019) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.0001, 'degree': 3}
0.950 (+/-0.019) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.001, 'degree': 3}
0.950 (+/-0.019) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.01, 'degree': 3}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.0001, 'degree': 4}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.001, 'degree': 4}
0.950 (+/-0.018) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.01, 'degree': 4}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.0001, 'degree': 5}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.001, 'degree': 5}
0.944 (+/-0.022) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.01, 'degree': 5}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.0001, 'degree': 6}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.001, 'degree': 6}
0.943 (+/-0.019) for {'kernel': 'poly', 'C': 1e-08, 'gamma': 0.01, 'degree': 6}
```

K-Nearest Neighbour

In [6]:

```
#declaramos los parámetros que vamos a utilizar

parameters_knn = {'n_neighbors': [1,10,20,30,40,50,60,70,80,90,100]}

clf_knn = neighbors.KneighborsClassifier()
tunning_knn = GridSearchCV(clf_knn, parameters_knn, cv = 5)
tunning_knn.fit(data, numb_label)

print "Best parameters set found on development set:"
print tunning_svm.best_params_

print ""

print "With score:"
print tunning_svm.best_score_

print ""

print "Total Scores:"
print ""
means = tunning_svm.cv_results_['mean_test_score']
stds = tunning_svm.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, tunning_svm.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

Out [6]:

```
Best parameters set found on development set:
{'n_neighbors': 1}

With score:
0.9294790343074968

Total Scores:

0.929 (+/-0.004) for {'n_neighbors': 1}
0.832 (+/-0.001) for {'n_neighbors': 10}
0.774 (+/-0.012) for {'n_neighbors': 20}
0.732 (+/-0.044) for {'n_neighbors': 30}
0.689 (+/-0.042) for {'n_neighbors': 40}
0.670 (+/-0.035) for {'n_neighbors': 50}
0.635 (+/-0.063) for {'n_neighbors': 60}
0.614 (+/-0.060) for {'n_neighbors': 70}
0.601 (+/-0.039) for {'n_neighbors': 80}
0.591 (+/-0.033) for {'n_neighbors': 90}
0.574 (+/-0.023) for {'n_neighbors': 100}
```

Multi-Layer Perceptron

In [7]:

```
#declaramos los parámetros que vamos a utilizar

parameters_mlp = {'hidden_layer_sizes': [100,200,300,400,500,600,700,800],
                  'activation': ['relu','logistic','tanh']}

clf_mlp = MLPClassifier()

tunning_mlp = GridSearchCV(clf_mlp, parameters_mlp, cv = 5)
tunning_mlp.fit(data, numb_label)

print "Best parameters set found on development set:"
print tunning_svm.best_params_

print ""

print "With score:"
print tunning_svm.best_score_

print ""

print "Total Scores:"
print ""
means = tunning_svm.cv_results_['mean_test_score']
stds = tunning_svm.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, tunning_svm.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

Out [7]:

```
Best parameters set found on development set:
{'activation': 'relu', 'hidden_layer_sizes': 1200}

With score:
0.9015247776365947

Total Scores:

0.872 (+/-0.022) for {'activation': 'relu', 'hidden_layer_sizes': 100}
0.868 (+/-0.031) for {'activation': 'relu', 'hidden_layer_sizes': 200}
0.874 (+/-0.023) for {'activation': 'relu', 'hidden_layer_sizes': 300}
0.881 (+/-0.049) for {'activation': 'relu', 'hidden_layer_sizes': 400}
0.874 (+/-0.044) for {'activation': 'relu', 'hidden_layer_sizes': 500}
0.890 (+/-0.022) for {'activation': 'relu', 'hidden_layer_sizes': 600}
0.882 (+/-0.021) for {'activation': 'relu', 'hidden_layer_sizes': 700}
0.887 (+/-0.029) for {'activation': 'relu', 'hidden_layer_sizes': 800}
0.891 (+/-0.027) for {'activation': 'relu', 'hidden_layer_sizes': 900}
0.900 (+/-0.024) for {'activation': 'relu', 'hidden_layer_sizes': 1000}
0.888 (+/-0.020) for {'activation': 'relu', 'hidden_layer_sizes': 1100}
0.902 (+/-0.014) for {'activation': 'relu', 'hidden_layer_sizes': 1200}
0.886 (+/-0.036) for {'activation': 'relu', 'hidden_layer_sizes': 1300}
0.893 (+/-0.024) for {'activation': 'relu', 'hidden_layer_sizes': 1400}
0.889 (+/-0.010) for {'activation': 'relu', 'hidden_layer_sizes': 1500}
0.750 (+/-0.054) for {'activation': 'logistic', 'hidden_layer_sizes': 100}
0.794 (+/-0.036) for {'activation': 'logistic', 'hidden_layer_sizes': 200}
0.803 (+/-0.049) for {'activation': 'logistic', 'hidden_layer_sizes': 300}
```

```
0.816 (+/-0.034) for {'activation': 'logistic', 'hidden_layer_sizes': 400}
0.825 (+/-0.044) for {'activation': 'logistic', 'hidden_layer_sizes': 500}
0.819 (+/-0.049) for {'activation': 'logistic', 'hidden_layer_sizes': 600}
0.818 (+/-0.058) for {'activation': 'logistic', 'hidden_layer_sizes': 700}
0.818 (+/-0.025) for {'activation': 'logistic', 'hidden_layer_sizes': 800}
0.821 (+/-0.030) for {'activation': 'logistic', 'hidden_layer_sizes': 900}
0.834 (+/-0.048) for {'activation': 'logistic', 'hidden_layer_sizes': 1000}
0.828 (+/-0.030) for {'activation': 'logistic', 'hidden_layer_sizes': 1100}
0.828 (+/-0.038) for {'activation': 'logistic', 'hidden_layer_sizes': 1200}
0.835 (+/-0.045) for {'activation': 'logistic', 'hidden_layer_sizes': 1300}
0.821 (+/-0.030) for {'activation': 'logistic', 'hidden_layer_sizes': 1400}
0.833 (+/-0.053) for {'activation': 'logistic', 'hidden_layer_sizes': 1500}
0.638 (+/-0.038) for {'activation': 'tanh', 'hidden_layer_sizes': 100}
0.713 (+/-0.059) for {'activation': 'tanh', 'hidden_layer_sizes': 200}
0.706 (+/-0.027) for {'activation': 'tanh', 'hidden_layer_sizes': 300}
0.721 (+/-0.035) for {'activation': 'tanh', 'hidden_layer_sizes': 400}
0.736 (+/-0.037) for {'activation': 'tanh', 'hidden_layer_sizes': 500}
0.755 (+/-0.086) for {'activation': 'tanh', 'hidden_layer_sizes': 600}
0.759 (+/-0.038) for {'activation': 'tanh', 'hidden_layer_sizes': 700}
0.768 (+/-0.021) for {'activation': 'tanh', 'hidden_layer_sizes': 800}
0.749 (+/-0.041) for {'activation': 'tanh', 'hidden_layer_sizes': 900}
0.762 (+/-0.041) for {'activation': 'tanh', 'hidden_layer_sizes': 1000}
0.765 (+/-0.059) for {'activation': 'tanh', 'hidden_layer_sizes': 1100}
0.774 (+/-0.029) for {'activation': 'tanh', 'hidden_layer_sizes': 1200}
0.774 (+/-0.020) for {'activation': 'tanh', 'hidden_layer_sizes': 1300}
0.767 (+/-0.021) for {'activation': 'tanh', 'hidden_layer_sizes': 1400}
0.764 (+/-0.049) for {'activation': 'tanh', 'hidden_layer_sizes': 1500}
```

Generar e imprimir matrices de confusión

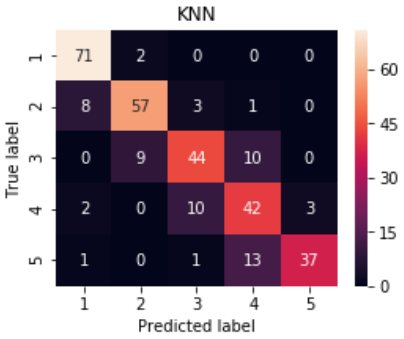
In [8]:

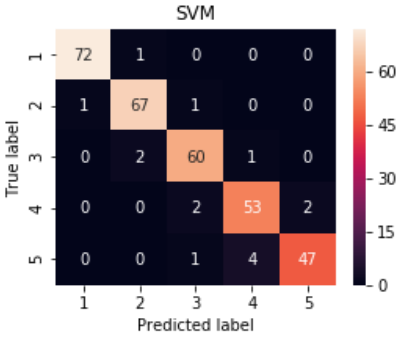
```
#seleccionamos los parametros del clasificador
clf_knn = neighbors.KNeighborsClassifier(20)
clf_svm = svm.SVC(kernel='poly', C=1e-12, gamma=0.01, degree=3)
clf_rf = ensemble.RandomForestClassifier(n_estimators = 75, criterion = 'entropy',
                                         max_depth = 12)
clf_mlp = MLPClassifier(solver='adam', hidden_layer_sizes = 1200, activation = 'relu')

ratio = len(image_list)*0.8
ratio = int(ratio)
print "\nTraining images: ",ratio
print "Test images: ",len(image_list)-ratio

#creamos las variables de entrenamiento cogiendo un 80% del total de datos
X_train,y_train = data[:ratio], numb_label[:ratio]
X_test,y_test = data[ratio+1:], numb_label[ratio+1:]

#entrenamos el clasificador
y_pred_knn = clf_knn.fit(X_train,y_train).predict(X_test)
y_pred_svm = clf_svm.fit(X_train,y_train).predict(X_test)
y_pred_rf = clf_rf.fit(X_train,y_train).predict(X_test)
y_pred_mlp = clf_mlp.fit(X_train,y_train).predict(X_test)

#generamos la confusion matrix
cm_knn = confusion_matrix(y_test, y_pred_knn)
cm_svm = confusion_matrix(y_test, y_pred_svm)
cm_rf = confusion_matrix(y_test, y_pred_rf)
cm_mlp = confusion_matrix(y_test, y_pred_mlp)

print "\nScore KNN: ",clf_knn.score(X_test,y_test)*100,"%\n"
print clf_knn
print "\nScore SVM: ",clf_svm.score(X_test,y_test)*100,"%\n"
print clf_svm
print "\nScore Random Forest: ",clf_rf.score(X_test,y_test)*100,"%\n"
print clf_rf
print "\nScore Multi Layer Perceptron: ",clf_mlp.score(X_test,y_test)*100,"%\n"
print clf_mlp

plt.figure(figsize = (4,3))
df_cm = pd.DataFrame(cm_knn, index = [i for i in "12345"], columns = [i for i in "12345"])
sn.heatmap(df_cm, annot=True)
plt.title('KNN')
plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.figure(figsize = (4,3))
df_cm = pd.DataFrame(cm_svm, index = [i for i in "12345"], columns = [i for i in "12345"])
sn.heatmap(df_cm, annot=True)
plt.title('SVM')
```

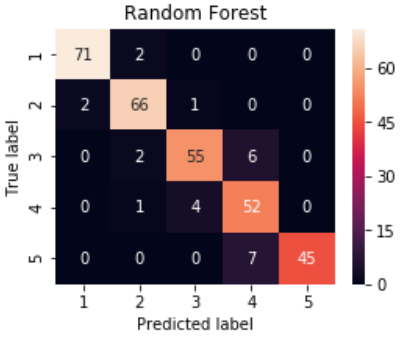
```
plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.figure(figsize = (4,3))
df_cm = pd.DataFrame(cm_rf, index = [i for i in "12345"], columns = [i for i in "12345"])
sn.heatmap(df_cm, annot=True)
plt.title('Random Forest')
plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.figure(figsize = (4,3))
df_cm = pd.DataFrame(cm_mlp, index = [i for i in "12345"], columns = [i for i in "12345"])
sn.heatmap(df_cm, annot=True)
plt.title('Multi Layer Perceptron')
plt.ylabel('True label')
plt.xlabel('Predicted label')

cv2.waitKey(0)
cv2.destroyAllWindows()
```

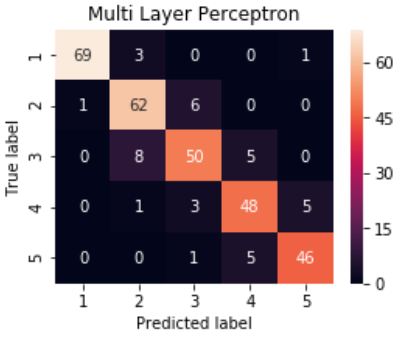
Out [8]:

```
Training images:  1259
Test images:  315

Score KNN:  79.93630573248409 %

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=20, p=2,
           weights='uniform')

Score SVM:  95.22292993630573 %

SVC(C=1e-12, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='poly',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Score Random Forest:  92.03821656050955 %

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=12, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=75, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

Score Multi Layer Perceptron:  87.57961783439491 %

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=1200, learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```


KNN
True label
Predicted label
71 2 0 0 0
8 57 3 1 0
0 9 44 10 0
2 0 10 42 3
1 0 1 13 37
1 2 3 4 5
60 45 30 15 0


SVM
True label
Predicted label
72 1 0 0 0
1 67 1 0 0
0 2 60 1 0
0 0 2 53 2
0 0 1 4 47
1 2 3 4 5
60 45 30 15 0


Random Forest
True label
Predicted label
71 2 0 0 0
2 66 1 0 0
0 2 55 6 0
0 1 4 52 0
0 0 0 7 45
1 2 3 4 5
60 45 30 15 0


Multi Layer Perceptron
True label
Predicted label
69 3 0 0 1
1 62 6 0 0
0 8 50 5 0
0 1 3 48 5
0 0 1 5 46
1 2 3 4 5
60 45 30 15 0

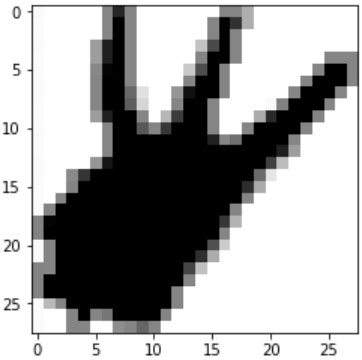
Ejemplo rápido

In [9]:

```
clf_knn = neighbors.KNeighborsClassifier(10)
clf_svm = svm.SVC(kernel='linear')
clf_rf = ensemble.RandomForestClassifier(20)
clf_mlp = MLPClassifier()

X_train,y_train = data[:ratio], numb_label[:ratio]
X_test,y_test = data[ratio+1:], numb_label[ratio+1:]

#entrenamos el clasificador
y_pred_knn = clf_knn.fit(X_train,y_train).predict(X_test)
y_pred_svm = clf_svm.fit(X_train,y_train).predict(X_test)
y_pred_rf = clf_rf.fit(X_train,y_train).predict(X_test)
y_pred_mlp = clf_mlp.fit(X_train,y_train).predict(X_test)

example=ratio+1

#hacemos la prediccion sobre el ultimo valor

print 'Prediction SVM',clf_svm.predict(data[[example]])
print 'Prediction Random Forest',clf_rf.predict(data[[example]])
print 'Prediction KNN',clf_knn.predict(data[[example]])
print 'Prediction Multi Layer Perceptron',clf_mlp.predict(data[[example]])
print 'Real class: ',numb_label[example]

#mostramos por pantalla
plt.imshow(images[example],cmap=plt.cm.gray_r, interpolation="nearest")
plt.show()
```

Out [9]:

```
Prediction SVM [3]
Prediction Random Forest [3]
Prediction KNN [3]
Prediction Multi Layer Perceptron [3]
Real class:  3
```



Animación en vivo

In [10]:

```
# Declaramos los distintos clasificadores que vamos a utilizar
clf_knn = neighbors.KNeighborsClassifier(1)
clf_svm = svm.SVC(kernel='poly', C=1e-12, gamma=0.1, degree = 3)
clf_rf = ensemble.RandomForestClassifier(n_estimators = 75, criterion = 'entropy',
                                         max_depth = 12)
clf_mlp = MLPClassifier(hidden_layer_sizes = 1200, activation = 'relu')

#Cargamos el 100% de los datos en las variables de entrenamientos,
#ya que para este caso no tenemos test
X_train,y_train = data[:], numb_label[:]

im_size = 28

#Entrenamos nuestros clasificadores
clf_knn.fit(X_train, y_train)
clf_svm.fit(X_train, y_train)
clf_rf.fit(X_train, y_train)
clf_mlp.fit(X_train, y_train)

#Declaracion variables utilizadas en la caputura de video y en la ROI
cap_depth = cv2.VideoCapture(1)
cap_rbg = cv2.VideoCapture(0)
x,y,w,h = 0,0,100,100

while True:

    ret, frame_depth = cap_depth.read()
    ret, frame_rgb = cap_rbg.read()

    #Hacemos efecto espejo
    frame_depth = cv2.flip(frame_depth,1)
    #Declaramos la mascara para quitarnos el fondo
    lower_green = np.array([0, 154, 0])
    upper_green = np.array([0, 154, 0])
    mask = cv2.inRange(frame_depth, lower_green, upper_green)
    #Aplicamos un filtro para quitarnos un poco de ruido
    median = cv2.medianBlur(mask ,15)
    #Invertimos los colores de la imagen.
    #Esto sirve para que funcione bien la extraccion de los contornos
    median = 255 - median;
    #Reducimos el tamaño de la imagen para reducir coste computacional
    resized_image = cv2.resize(median, (200, 150))

    #Buscamos los contornos en la imagen
    median, contours, hierarchy = cv2.findContours(median, cv2.RETR_TREE,
                                                   cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) != 0:
```

```
    # Dibujamos los contornos
    cv2.drawContours(median, contours, -1, (122,122,0), 3)

    #Buscamos el contorno de area maxima
    c = max(contours, key = cv2.contourArea)

    #Declaramos las medidas del rectangulo de contorno de area maxima
    x,y,w,h = cv2.boundingRect(c)

    #Dibujamos el contorno
    cv2.rectangle(median,(x,y),(x+w,y+h),(255,255,255),2)

#Guardamos en la variable roi_max la imagen que se encuentra dentro de la ROI
roi_max = median[y:y+h,x:x+w]

#Volvemos a reducir el tamaño de la imagen
resized_roi = cv2.resize(roi_max, (im_size, im_size))

#Convertimos las imagenes 2D en 1D y las guardamos en data
nx, ny = resized_roi.shape
data = resized_roi.reshape(nx*ny)
data = resized_roi.reshape(1, -1)

#Imprimir los resultados en video
result_knn = clf_knn.predict(data)
result_svm = clf_svm.predict(data)
result_rf = clf_rf.predict(data)
result_mlp = clf_mlp.predict(data)

cv2.putText(frame_rgb, str(result_knn), (10,40), cv2.FONT_HERSHEY_SIMPLEX, 1,
                (0,255,0),2)
cv2.putText(frame_rgb, str(result_svm), (80,40), cv2.FONT_HERSHEY_SIMPLEX, 1,
                (255,0,0),2)
cv2.putText(frame_rgb, str(result_rf), (150,40), cv2.FONT_HERSHEY_SIMPLEX, 1,
                (0,0,255),2)
cv2.putText(frame_rgb, str(result_mlp), (220,40), cv2.FONT_HERSHEY_SIMPLEX, 1,
        (255,255,0),2)
cv2.putText(frame_rgb, "KNN", (10,310), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0),2)
cv2.putText(frame_rgb, "SVM", (10,350), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0),2)
cv2.putText(frame_rgb, "Random Forest", (10,390), cv2.FONT_HERSHEY_SIMPLEX, 1,
        (0,0,255),2)
cv2.putText(frame_rgb, "MLP", (10,430), cv2.FONT_HERSHEY_SIMPLEX, 1,
                (255,255,0),2)
cv2.putText(frame_rgb, "Press Q to exit", (390,470), cv2.FONT_HERSHEY_SIMPLEX, 1,
        (255,255,255),2)

#Mostramos el frame filtrado y el RGB
cv2.imshow('frame_rgb', frame_rgb)
cv2.imshow('median', median)

#salimos de la ejecucion con q
if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
        break

cap_depth.release()
cap_rbg.release()
cv2.destroyAllWindows()
cv2.waitKey(0)
```

Out [10]:

```
-1
```